UTOPIA

Antonio Sapuppo
Beatriz Fisac Lozano
Daniel Aguilera Trigos
Ruben Campo Alonso
Supervisor: Rene Hansen

7th semester
September 2007 - December 2007

Contents

1 UTOPIA **4**
- 1.1 Introduction and Motivation . . . 4
- 1.2 Possible Solutions . . . 5
  - 1.2.1 Stationary Points . . . 5
  - 1.2.2 Flood techniques . . . 7
  - 1.2.3 Link state routing algorithms . . . 8
- 1.3 What is Utopia? . . . 9

2 Technology evaluation **11**
- 2.1 Technical Analysis . . . 11
  - 2.1.1 Wireless Technologies . . . 11
  - 2.1.2 Development Patform . . . 13
- 2.2 In depth Analysis . . . 14
  - 2.2.1 Bluetooth . . . 14
  - 2.2.2 Java 2 Micro Edtion . . . 16
- 2.3 P2P context . . . 22
  - 2.3.1 Definition . . . 22
  - 2.3.2 Types of p2p networks . . . 22
  - 2.3.3 P2P characteristics . . . 27

3 Design **29**
- 3.1 Possible scenarios . . . 29
  - 3.1.1 Direct message to neighbors . . . 30
  - 3.1.2 Send message through one or more intermediaries and get the confirmation message . . . 31
  - 3.1.3 Send message through one or more intermediaries and loose the confirmation message . . . 32
  - 3.1.4 Destination node not reachable . . . 33
- 3.2 Use cases . . . 34
- 3.3 Class diagram . . . 35

4 Architecture **39**
- 4.1 Server . . . 39
- 4.2 Client . . . 45
  - 4.2.1 Device discovery . . . 45
  - 4.2.2 Service Discovery . . . 49
  - 4.2.3 Connect to a service . . . 52

5 Utopia algorithm **54**
5.1 Type of the node 54
5.2 Structure of the database 56
5.3 Neighbors table 56
5.4 Size of network 57
5.5 Structure of the message 57
5.5.1 Two way handshakes 60
5.5.2 Message state transition diagram 62
5.6 Protocol 63
5.6.1 Source node protocol 63
5.6.2 Forward node protocol 64
5.6.3 Waiting Timer 65
5.6.4 Number of neighbors 67
5.6.5 Neighbor selection protocol 72

6 Application **75**
6.1 User manual 75
6.1.1 Requirements 75
6.1.2 Actions 78
6.2 Tests 85
6.2.1 Fast device discovery 87
6.2.2 Forward message with one intermediary node 88
6.2.3 Forward a message with several intermediariers node . . . 88
6.2.4 Concurrent reception of messages 89
6.2.5 Modifications in the network (add, modify and remove nodes) 90

7 Conclusions and future work **93**

8 Ackowledgements **95**

Bibliography **95**

List of Figures **98**

Abstract Mobile ad hoc networks enable communication between nodes by creating connection over several intermediate hops, routing and topology control over such networks is a difficult task due the high dynamics of the network, Utopia proposes an advance broadcast algorithm over this kind of networks that will allow overcoming some of Bluetooth limitations, more specifically range. Utopia makes use of incremental flood methods and introduces new techniques to save time and resources while executing this kind of algorithms. As a result will be possible to message devices separated long distances by making use of intermediate Bluetooth devices.

Chapter 1

UTOPIA

1.1 Introduction and Motivation

There are many ways to connect different devices, from cable based systems to wireless enable like WiFi, Bluetooth or even Wibree, but more important than that is what you can do with them. Bluetooth originally designed as a more effective replacement for IrDA is pretty flexible but even though has some limitations like range.

Mobile ad hoc networks enable communication between nodes by creating connection over several intermediate hops; reactive routing algorithms are able to control high dynamic network topologies and almost arbitrary network sizes.

Topology control in ad-hoc networks has been an area of active research and many techniques to improve capacity and high dynamic adaptation have been investigated, however maintaining a full topology control inside an ad-hoc Bluetooth high dynamic network is a task really difficult to accomplish since nodes inside the network can change of position too fast, leave or join at anytime or simple crash, this kind of scenario is common in real life environments.

Peer to peer philosophy is comparable with this approach but without requiring a stationary network infrastructure, acting independently from the underlying network infrastructure and creating a free, decentralized, self organizing network.

Common p2p algorithms have been optimized to find information within their overlay network, but for information exchange normally rely on TCP or similar protocols that assume stable connections and where information exchange is achieve by creating a direct connection between source and destination; this kind of approach is impossible inside a Bluetooth ad-hoc network since information needs to travel through various nodes or hops along the net.

Our research emphasis in developing p2p information sharing environments for Bluetooth enables devices. In particular we focus in covering large physical areas throw various p2p nodes connected in an ad-hoc manner; in order to

achieve such a task various p2p algorithms are proposed with the objective of ensuring, lower delay, best effort delivery and lesser number of hops; most of them behave in an intelligent manner taking into account different factors like type of node, class and speed. Such algorithms and techniques are implemented in a controlled environment, and their performance, behavior and limitations are discussed.

Different types of flooding or broadcasting techniques exist; each one of them has some advantages and fallbacks; the most common approach is a simple flood technique which broadcast messages to all the neighbors that a node has, and each one of them does the same process again until destination is found. A more intelligent approach is to increase gradually the number of successors; in a similar way the *deph* of each broadcast could also be increased. As a result lower delay, less overhead and more reliable delivery of messages is achieved.

Over the next pages we'll deal with the creation of a draft technology (Utopia) that will allow us to overcome some of Bluetooth limitations, more specifically range by making use of an incremental flood techniques and introducing stochastic random methods. As a result you will be able to message devices within the maximum range of Bluetooth but also devices that might be hundreds meters away, only limited by the number of Bluetooth nodes participating in the network. For example, new exciting and innovating applications like supporting military operations may be possible since no or very little on site management is required, in a similar way creating new urban perceptions, experiences and social interactions with cell phones, notebooks, PDA or any Bluetooth enable device could be possible.

1.2 Possible Solutions

This section explain possible solutions for creating a Bluetooth ad-hoc network capable to manage high mobility of nodes and therefore expanding Bluetooth range, a simplistic approach is taken and only an overview of the solutions is shown. It includes the following topics:

- Stationary Point
- Flood techniques
- Link state routing algorithms.

1.2.1 Stationary Points

In this case the problem of the device´s mobility is attacked by having some nodes that never move this work better when static Bluetooth devices have a range bigger or equal to 100 meters (class 1 Bluetooth devices); consequently one stationary point can manage one big area and supervise all the devices in their zone.

One of these ideas is related with stationary points; in this case the problem of the device´s mobility is managed because we have some immobile nodes. This idea seemed better when we saw that the Bluetooth devices can have a range of until 100 meters if they belong to Class 1, so only one stationary point could manage one area and supervise all the devices in this zone.

Figure 1.1 shows a possible stationary point environment.

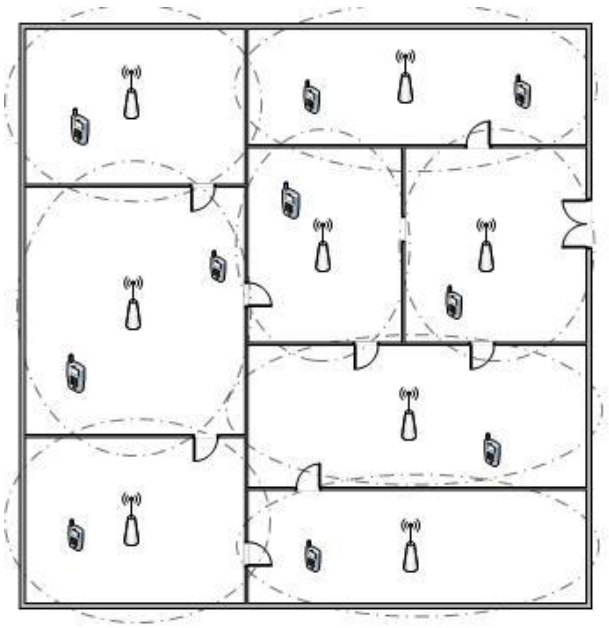


Figure 1.1: Distribution of stationary points

This kind of approach offers a lot of possible interesting features, like the possibility of knowing where a specific device is (BT location services) and send messages to a group of users in the same area between others.

Sending of messages only depends on stationary points; therefore routing process is simplified since only static nodes forward messages. This means that one message travels only from the source to the destination throw stationary points.

One of the disadvantages of this environment is that it is very similar to Wi-Fi or even worst Wi-Fi already solves this problem in a better way leaving nothing new or innovation possibilities compared to a stationary point solution. Another pitfall is that the challenge of building a total mobile network on the fly is not achieved by this technique; deployment and planning of the infrastructure need to be done first.

Although we did not choose this approach, some interesting ideas were taken from it like partition the network by type of nodes, the inspiration was that for

instance printers could act as stationary points since it's not likely that a printer move of place.

1.2.2 Flood techniques

This strategy is simple; each node that receives a unique (not already received) message repeats the message to all of its neighbors, except the neighbor it received the message from. Each message has a time to live (ttl) counter the value is decremented by each node as it relays the message, once the ttl counter reaches zero the message is destroyed. Significant network traffic may be generated due to broadcast nature of this technique.

One disadvantage of this approach is that it does not scale well (O (n2)), the bandwidth needed increase exponentially as the number of nodes grows, raising the number of participating devices will cause the network to quickly reach bandwidth saturation. On the other hand it is simple to design and efficient since it highly increases the chances of successfully delivering a message. As long as the network size is not too big this approach can perform efficiently.

Gnutella is a living example of a simple flooding mechanism, when a peer makes a query; the query is then broadcasted to all the neighbor peers. If its neighbor peers could not solve the query, then the query is broadcasted to neighbor's neighbors peers and so on [1].

Different variations of broadcasting messages exist, the most common and popular are a modified Breadth First Search (BFS), iterative depending and random walks.

In a modified BFS nodes randomly choose only a portion of their neighbors to forward the message to, this reduces the average message production but still generates a large overhead.

Iterative depending methods searches the destination at increasing depths, these algorithms achieve best results when it is possible to find our target at a small depth and in our case increase the chances of finding the destination.

Finally in random walks, the requesting node sends out k messages to an equal number of randomly chosen neighbors. Each of these messages follows its own path, having intermediate nodes forward it to a randomly chosen neighbor at each step, TTL methods are used to stop message propagation to infinity. The algorithm achieves some local load balancing, since no nodes are favored in the forwarding process over others. On the other hand success highly depends on network topology and the random choices made [2].

Figure 1.2 shows a simple message propagation scenario using flooding techniques.

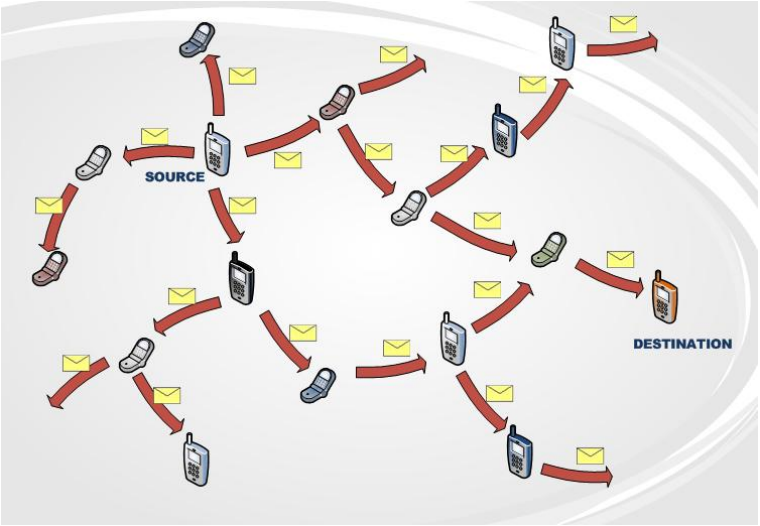


Figure 1.2: Flood scenario

Flooding techniques are important for Utopia because interesting concepts were taken from them, like using random methods to introduce some local load balancing, as a matter of fact different variations of these techniques are used inside of our final proposed algorithm, and a total flooding scenario is taken when everything else fails.

1.2.3 Link state routing algorithms

Routing in mobile ad-hoc networks with a large number of nodes or with high mobility is a very difficult task and current routing protocols do not really scale well with these scenarios [3]. There exist three basic routing techniques when talking about networks, proactive, reactive and a hybrid approach.

Proactive routing keep routing information current at all times, it is good for static networks but in our case is useless since we are dealing with a high dynamic network; in the other hand reactive routing find a route to the destination only when a request comes in, this kind of algorithms perform better on dynamic networks; finally, hybrid techniques keeps only some information current.

The most likeable solution for maintaining routing operations inside a high dynamic Bluetooth ad-hoc network is using reactive routing algorithms since they perform better inside dynamic environments.

A simple reactive routing algorithm inside our Bluetooth ad-hoc network works as follows, when a message needs to be sent, routing tables are queried

to check if there exist a route to the destination, if sources do not know a path to destination a discovery request is issued; such request consist of flooding the network to find a route to the destination. Since Bluetooth devices tend to move faster than our ability to track them maintaining routing tables is a naive approach because routing information will be outdated soon.

A similar approach is a link state routing algorithm and it works generally like this: each Bluetooth device starts by sensing the state of the links with its neighbors; this process is done by periodically doing a research discovery process. Then this information is flooded to all the other nodes, finally each one builds a database containing the topology of the entire network. The database is kept up to date by using the same flooding mechanism when changes in a link are detected and also periodically. Each node contains now enough information at any time to build a view of the entire network and send messages [4].

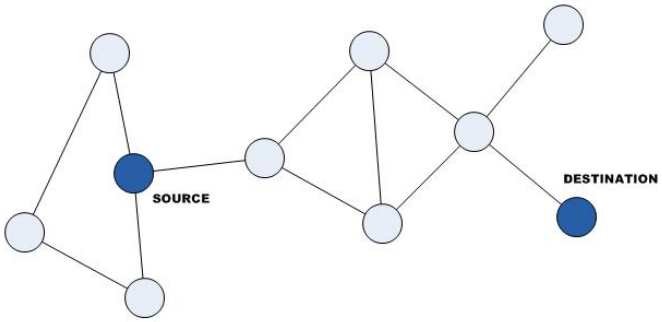


Figure 1.3: Possible network structure

Figure 1.3 shows a possible network topology for a Bluetooth ad-hoc network.

Even though we didn't choose these techniques they could solve the problem for small size of networks. Some interesting ideas were taken from routing protocols, one of them is the flooding technique used by some reactive routing algorithms when discovering a new route to destination.

1.3 What is Utopia?

Utopia is the main case of study; we focus our research and effort on it. Utopia is a technology that aims to solve the problem of extending Bluetooth range by creating an ad-hoc network between several Bluetooth devices, attacks problems like high mobility of nodes (high dynamic networks), all time system availability and devices failures at any time.

Utopia´s main engine is an intelligent iterative depending flood technique that makes use of several different concepts of flooding, from introducing ran-

dom decisions to probabilistic estimations base on the ability of a device to perform better in a given situation. It also tries to ensure best effort delivery of information by defining a simple message distribution protocol.

Cutting edge technologies surround this engine; one of the most important ones is J2ME since Utopias functionality is demonstrated by implementing a distributed chat application over it. It's important to mention that Utopia´s technology is in a non mature state now.

Algorithm and protocols are described in much more detail in future sections of this document, also some pitfalls of Utopia are discussed since this kind of technology can perform better in specific scenarios where some limitations of it doesn't interrupt systems functionality.

Chapter 2

Technology evaluation

2.1 Technical Analysis

In this section a technology evaluation of the resource used is given. At beginning it is going to be explained the reasons for choosing particular kind of resources, following a in depth technology study of the resources chosen will be given and an overwiev of the P2P context will be described.

2.1.1 Wireless Technologies

The current section explains main reasons for choosing a particular kind of technology as a resource needed to set up Utopia algorithm enviroment.

An average mobile device user is always presented with a lot of different wireless technology options to choose from, below a comparison of mayor competing wireless technologies is shown, strengths and weakness of each one is analyzed.

Figure 2.1 shows wireless technogies comparison considering advantages and disadvantages of them [5] [6] [7].

	ADVANTAGES	DISAVANTAGES
WIFI	1. Freedom of movement 2. Permanent access to Internet. 3. Many reliable WIFI products on the market.	1. High power consumption. 2. Can get interference. 3. Open up your network to the public
IrDA	1. Available in many devices. 2. High bandwidth. 3. Low power consumption.	1. Direct sight between devices. 2. Very short range. 3. Impossibility to have networks
Bluetooth	1. Can go through walls. 2. Low energy consumption. 3. Many reliable Bluetooth products on the market. 4. Have simplex and duplex communication.	1. It is not as fast as IrDA. 2. Limited voice compatibility 3. Short range.

Figure 2.1: Wireless technologies comparison

Bluetooth is our favourite choice for several reasons; one of the most important is that in contrast with IrDA Bluetooth offers a bigger communication range and is not necessary to maintain a direct line of sight with the other communication device.

In contrast with WLAN, Bluetooth is more common in mobile phones and it can offer more services and more facilities to connect mobile phones.

Bluetooth is an innovative technology, provides a way to connect and exchange information between devices over a secure, globally unlicensed short-range radio frequency; unfortunately, Bluetooth does have some limitations.

2.1.2 Development Patform

The purpose of this section is to study and analyze which mobile programming languages are most appropriated to use in the implementation of our research and give a general overview of why we have decided to use one of them.

Three mayor competing languages are analyzed: Symbian c++, Python and Java 2 Micro Edition.

In Figure 2.2 a comparison between these three languages is given considering advantages and disadvantages of them [8] [9] [10].

	ADVANTAGES	DISADVANTAGES
Symbian C++	1.The most common operative system on mobile phones is based in this language. 2.Properties for less battery consumption and memory saving. 3.Powerful because it has better knowledge of the OS functionality. 4.It is fast because is executed as native compilation code.	1.There exist few environments or toolkits for this language. 2.The development and the simulation need powerful computers. 3.Difficult programming language. 4. It needs a certificate so that the mobile phones accept the programs. 5. Difficult memory management (manual control).
Python	1.Allows the programmer write a wide variety of programs in a very easy way. 2.This language is free so there are not licenses or copyrights. 3.Script Language. This means that is a very productive language, more instructions in less space.	1. Very slowly execution time that cannot be solved in devices with little power. 2. Used in few mobile phones. 3. Limited access to the resources.
J2ME	1.It is the most well known so there exists a lot of material about it. 2.The most of mobile phones use Java. 3.There exists previous knowledge in this language. 4.The programs are very portable because they are executed on virtual machines.	1. It has a slow execution. 2. Limited access to the resources.

Figure 2.2: Develop platforms comparison

Having a quick look at Figure 2.2 we obviously see that one important advantage of Symbian C++ over the others languages that offers more possibilities and that it is more efficient language than the others. We could think that Symbian is the best solution but another aspect has to be considered: Symbian is used by many mobile phone manufacturers as their operating system, but Java 2 Micro Edition is used by even more as an application platform.

Finally, there is another important reason to use Java: it offers a great portability and for that, the most of mobile phones and even other devices use Java in their applications. Therefore our application could be extended by using this language.

2.2 In depth Analysis

The previous section presented reasons for choosing particular type of resources which are used on this project, the following section focuses on giving a theoretical overview and presenting the technical details of the resources.

2.2.1 Bluetooth

Bluetooth is a technology which possibilities the wireless connectivity between devices in a short distance. Bluetooth had an important evolution in few years and it has a large way to cover yet.

This section explains the main Bluetooth features. How we are going to use them in our application.

The physical layer operates in the unlicensed ISM (Industrial, Scientific and Medical) band at 2.45 GHz (this frequency is used for industrial, scientific and medical devices and is called ISM). In this frequency there exist a lot of devices that could interference in the Bluetooth transmission, so the transceiver system employs some techniques to avoid the interferences. The bit rate that can support this technology is in the range from 1Mbps until 2 or 3Mbps in the Enhanced Data Rate [11].

Different kind of Bluetooth class exists, they depends on the Radio Layer of the Bluetooth protocol stack which is places at the bottom of it. The Radio Layer defines the physical characteristic of the Bluetooth device - the range of Bluetooth depends on the power class of the radio. Figure 2.3 shows these classes [6].

	Max Power (mW)	Max Power (dBm)	Range (aprox.)
Class 1	100	20	100
Class 2	2.5	4	20
Class 3	1	0	1

Figure 2.3: Classes of Bluetooth chips

For example using a class 1 device, we could greatly increment the possibilities of our application, but it is more expensive than others and not intended for mobile phones.

Bluetooth has a lot of solutions to guarantee the security in the communication between devices. Some examples are the followings [5]:

- One technique of multiplexity called Spread Spectrum Frequency Hopping based on hopes over the frequency baseband. This is used to avoid the interferences with other applications and blocking the communication.

- Specifications for authenticate devices that try to connect to the Bluetooth network.
- Techniques for encode some aspects of information.

Bluetooth has the property of establish networks. Figure 2.4 shows the possible Bluetooth connection scenarios.

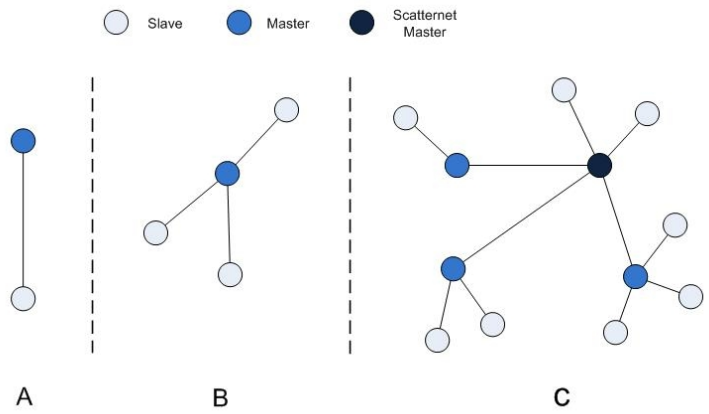


Figure 2.4: Bluetooth piconet and scatternet scenarios

Figure 2.4-a shows a Point-to-point connection between two devices. Master is the device which starts the connection and the other device is a slave. A master can be connected with one or more slaves. Figure 2.4-b shows a scenario where a master is connected with three slaves - this scenario is called Point-to-multipoint.

Figure 2.4-a and Figure 2.4-b show a Bluetooth network which is called piconet.

One device can be connected with one or more piconets at the same time: this scenario is called scatternet and it is shown in Figure 2.4-c.

In a piconet only a device can be the master and all the other devices are slaves. This is the same also for the scatternet scenario, in fact if a device partecipates in two different piconets, it can be both master and slave [11].

2.2.2 Java 2 Micro Edtion

Sun has developed different solutions for each field of technology during the last few years. Those editions are Java 2 Platform Enterprise Edition (J2EE) oriented to the business field, Java 2 Platform Standard Edition (J2SE) for the familiar and well-established desktop computer market and Java 2 Platform Micro Edition (J2ME) oriented to the programming with small devices. Each edition is oriented to very different application field [12].

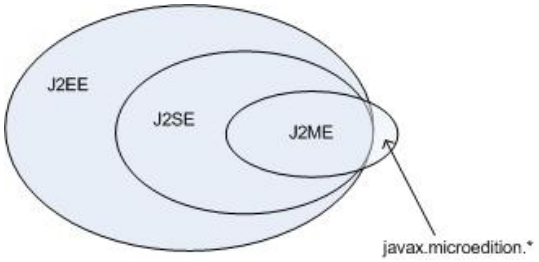


Figure 2.5: Java's family

Sun decided to separate these technologies due to efficiency reasons. J2ME is a subset of J2SE because this one has all the features of J2ME (excepting the javax.microedition packet) and also J2SE is a subset of J2EE as it is shown in Figure 2.5.

Figure 2.6 shows examples of devices for each edition of Java.

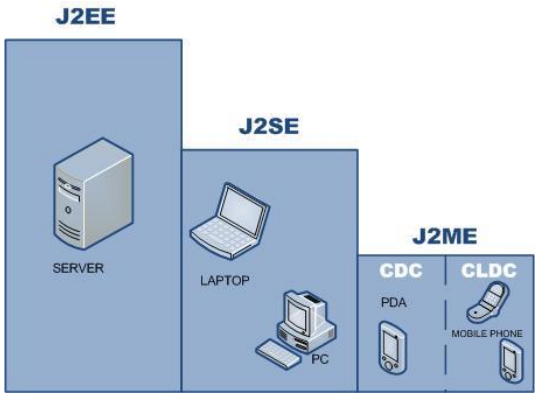


Figure 2.6: Examples of devices for each edition

J2ME components

Figure 2.7 shows the layers of Java 2 Micro Edtion architecture.

Figure 2.7: Layers of J2ME

The J2ME follows a level structure in which the highest levels use the capabilities of the lowest levels. As the rest of normal applications the Operating System execute and direct the application, the Virtual Machine converts the language in machine code. The main component of J2ME are configurations and profiles because give big flexibility to the platform. The object of dividing them is to adapt the virtual machine and the libraries for each group of devices. Configurations include a group of APIs obtaining the necessary to implement J2ME applications, and profiles extend the application capabilities [14].

Virtual Machine The Java Virtual Machine (JVM) is software that converts the Java intermediate language (bytecode) into machine code executable to the platform, communicates with the Operation System, observes the security rules and corrects the code.

J2ME has two Virtual Machines, KVM and CVM, each one is used for different configurations of J2ME. KVM is a special Virtual Machine that Sun developed for devices with special features like small memory; this is the case of the mobile phones and PDAs.

Configuration A configuration defines the basic J2ME runtime environment. The configuration describes the minimum set of APIs Java which allow develop basic applications, defining the Java features, features support for the Virtual

Machine and the basic Java libraries [14].

There are two configurations defined in J2ME: *Connected Device Configuration (CDC)* and the *Connected Limited Device Configuration (CLDC)* oriented to devices with limitations in memory and computational. CLDC is used with small devices with important restrictions in memory, power and network bandwidth like mobile phones, PDAs, etc. CDC is a superset of CLDC.

Figure 2.8 shows the Java Micro Edition configuration and how they are related between each others.

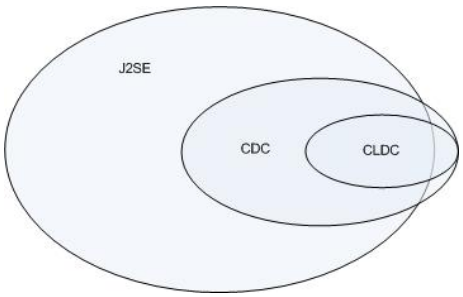


Figure 2.8: J2ME configurations

Figure 2.9 shows a comparison between CDC and CLDC configurations [12].

Connected Limited Device Configuration(CLDC)	Connected Device Configuration (CDC)
-Very simple user interface (including no UI at all). -Low level memory budgets (160 kB to 512 kB). -16 or 32 bit processors. -Wireless communication, possibly low bandwidth -Limited power, often battery operation. -Examples: Mobile phones, pagers and personal organizers.	-Large range of user interface capabilities (down to and including no UI). -Memory budgets in range of 2 to 16 megabytes. -16 or 32 bit processors. -Connectivity to some type of network. -Examples: TV set-top boxes, Internet TVs and high-end communicators.

Figure 2.9: Configuration characteristics

Profiles A profile is an APIs set intended for a specific application field, this APIs set contributes to the configuration with a specific functionality. Profiles identify devices with the functionality given (mobiles phones, etc) and the application kind executed in these. The profile extends a configuration, giving domain-specific classes to the core set of classes.

There are different profiles depended of the configuration chosen. CDC has Foudation, Personal, RMI profiles and CDLC configuration has PDA and Mobile Information Device Profile (MIDP).

MIDP is the profile in which Utopia framework is based, that's because it is oriented to devices with limited features, like small memory and low computation power [14].

After we have seen the components of J2ME it's easier to understand the full architecture shown in Figure 2.10.

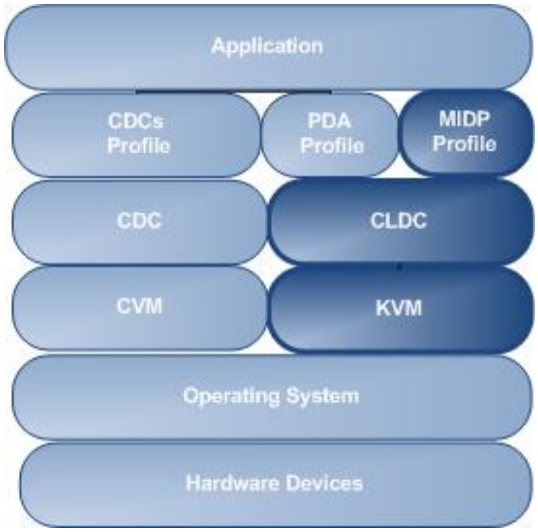


Figure 2.10: J2ME Architecture

MIDP MIDP is a key element of J2ME and combined with the Connected Limited Device Configuration (CLDC), provide a standard Java runtime environment for mobile information devices.

MIDP have a lot of benefits which are oriented to confront the limitations of small devices. Figure 2.11 shows some of these benefits [15].

BENEFITS	
User Interface Optimized	Adapt the User Interface to the small screen of the devices.
Extensive Connectivity	MIDP can support several connectivity standards like HTTP, datagrams, sockets, server socket and serial port.
Include Multimedia Applications	Give better control of graphics and input when they need it
Use and update applications over the air	The applications can be extend and update with better security and dynamically over the air.
Increment the security	MIDP give a more security model protecting the network, applications and mobile information devices.

Figure 2.11: Benefits of MIDP

Midlet is the name of the Java applications implemented using the profile MIDP, therefore a midlet is an application that can use the specification contribute by MIDP and CLDC. A midlet always is composed of a main class inherited from the *javax.microedition.midlet.MIDlet class* [14].

Figure 2.12 shows the packages of the MIDP API [15].

PACKAGES	
javax.microedition.midlet	Application Lifecycle package. Allow to interaction between the application and environment
javax.microedition.lcdui	Interface package. Give the features to implement User Interface.
javax.microedition.lcdui.game	Game package. Classes to built games for mobile devices.
javax.microedition.io	Core package. Network package. Support the networks following CLDC.
javax.microedition.rms	Persistence package. Allow to store data and then retrieve it.
java.lang	Core package. Fundamental classes of Java
java.util	Core package. Provide the date and time facilities.
javax.microedition.pki	Public Key package. Certificates are used to authenticate information for secure Connections.
javax.microedition.media	Audio Package. Compatible with Mobile Meida API giving multimedia features.
javax.microedition.media.control	Audio package. Define the control using in a Player.

Figure 2.12: Packages of MIDP API

Optional Package The top layer of Java 2 Micro Edition is composed by the optional package. These optional packages that J2ME can include extend the possibilities of Java including more APIs [13].

The only optional package used on Utopia is *Java APIs JSR 82 for Bluetooth*. The APIs Java for Bluetooth defines two packages which depend of package CLDC *javax.microedition.io* [25]:

- javax.bluetooth: it defines classes and basic interfaces for the discovery of devices and services, connection and communication. The communication across this package is low level: by the information flowing or by the transmission of arrays of bytes.
- javax.obex: it allows operating the high-level protocol OBEX. This protocol is very similar to HTTP and is used especially for the exchange of files.

2.3 P2P context

The following chapter introduces important p2p concepts and explains the relation between Utopia and such terms. First a formal definition of p2p networks is given, then types of different p2p networks are explained and finally general characteristics of p2p are related with this project.

2.3.1 Definition

The concept of peer to peer is relatable new; it was used before Internet as one to one communications, for example military operations during the Second World War II used peer to peer communications between two camps, each camp used a communication device and creates a direct link between both of them.

Even though p2p concept is not brand new, the way of using it has evolve since the use of networks like Internet, peer-to-peer with hyphens refers to the same concept but with the ability or characteristic where nodes (or peers) generates its own organization; any computer granted permission to become a node in the community is, at the same time, granted equality to every other node in the community. Adding peers to a network requires no re-organization, central or otherwise [16].

Accordingly with the popular online encyclopedia wiki, p2p can be defined as: *P2P overlay* network consists of all the participating peers as network nodes. There are links between any two nodes that know each other [18].

The following idea of a peer to peer network will clarify the use of this technique in our application. It is a communication model in which each node has the same capabilities and each one can initiate communication. In some cases like ours, this kind of networks are implementing by giving each node both server and client capabilities [17].

Peer to peer philosophy follows the idea in which all the members of a network have the same importance and can perform the same roles, better network reliability and flexibility is achieve by using and sharing resources of all members.

Utopia follows the same philosophy; when delivering a message it must travel through various Bluetooth devices, each one of them acts as an intermediary to deliver the message. Without the help of other devices such task could not be done, each device adds more resources to the overall network having as a consequence greater range for message delivery.

2.3.2 Types of p2p networks

It's possible to classify p2p networks into two major types, structured and unstructured, accordingly with how the nodes are linked within the overlay network. Unstructured networks can be subdivided at the same time into centralized, pure and hybrid.

Structured p2p networks employ a globally consistent protocol to ensure that any node can efficiently route a search to some peer that has specific content [18].

Every peer is responsible for specific content on the network. This kind of networks use distributed hash functions, assign values to each content and each peer on the network. After that run a global protocol to establish which node is responsible for a specific content. In this way, each time that a peer wants to find something, it use the global protocol to establish the peer or peers responsible for specific data and find them [19].

Utopia doesn't fall under structured p2p networks because of their ad hoc network and high dynamism, it's impossible to keep a consistent overlay network due to mobility of nodes and therefore running a protocol capable of assuring where to find specific content or nodes is not an intelligent approach. One of the most important characteristics of structured p2p networks is that overlay network is fixed, one luxury that Utopia is not blessed with.

Even thought structured p2p networks represent a power approach since finding and locating data can be assure at an specific point of time, they don't fit Utopias global environments. Unstructured p2p networks in the other hand match more closely Utopias requirements, in this kind of networks links are established arbitrarily, if a peer wants to find information in the network, the query has to be flooded through it in order to find peers and the requested data [20].

Most popular p2p networks are unstructured, Gnutella, Kazaa, eDonkey and Emule are examples of such networks. Centralized p2p systems where the first ones in appear, Napster is a good example of this kind of network. In this architecture exists one entity that contains all the information about the distribution of the network. This entity attends to the requests of the clients and gives them the necessary information to communicate with other nodes. This kind of network depends exclusively of the central entity with the disadvantage of representing a single point of failure [21].

Figure 2.13 shows centralized p2p network architecture, if central server fails system ability to find resources and nodes is completely lost.

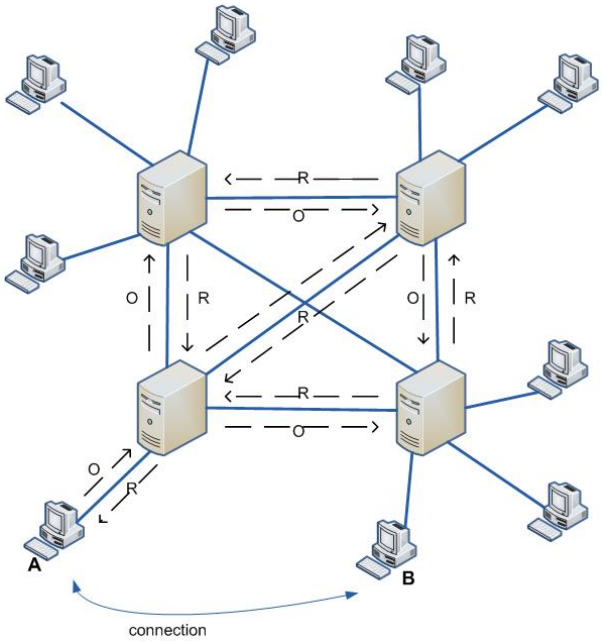


Figure 2.13: Centralized P2P network architecture

In Figure 2.13, node A is trying to find node B on the network, node A sends query (Q) to the central entity (central entity is constituted of four powerful servers working in a collaborative way only), query (Q) travels around all the collaborative servers and finally target B is found by the central entity, then central entity informs A of how to connect to B, finally a direct connection between A and B is established.

The central entity needs to be observed as a single unit that holds the knowledge of all the nodes connected to the network, the case where the central entity is only one server and all the other nodes query this server in order to know the location of other node or resources is the simplest one. Central entity can be as complex as required, but following the idea that when queried it can answered the location of any resource and node.

The main disadvantage of this architecture is that if central entity fails the whole network goes down, is mandatory that all nodes or peers make use of such entity.

Centralized architecture doesn't fit Utopias requirements since one of the main objectives of Utopia is to overcome Bluetooth range limitation, this means that most of the time no direct connection can be established between to nodes, it's necessary to use intermediary nodes to establish such connection.

The most important type of p2p network for Utopias algorithm is the one corresponding to pure p2p networks, in this kind of networks all nodes are peers, and each may function as router, client or server, according to the status of the query [16].

Utopias nodes can act as a server or client , when running Utopias protocols each node is given routing capabilities, this means that each one can take smart decisions of where to forward messages. Since Utopia carry out all of this functions can be almost classify as part of pure p2p networks.

Figure 2.14 shows common pure p2p architecture, no central entities are required to perform search of nodes or resources.

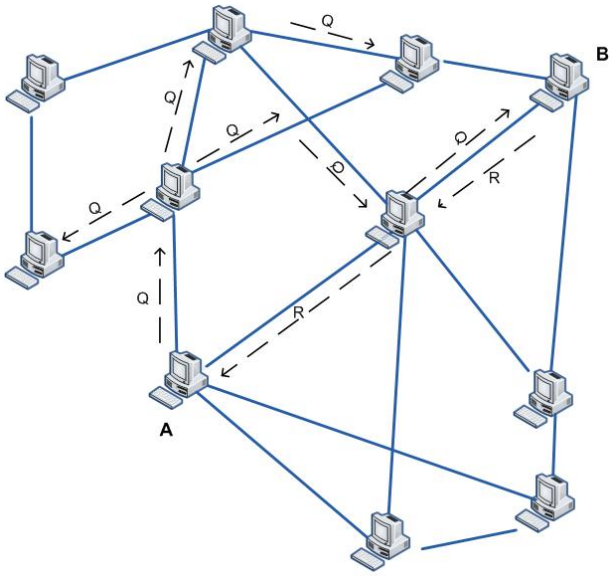


Figure 2.14: Pure P2P network architecture

In figure 2.14, node A is trying to find node B, node A sends a query to all the peers that are known by A, does nodes continuously forward the query Q in the same way that A did at the beginning, this process continuous until B is found, then when Query Q reaches B it answers the request using the intermediaries nodes, and finally a connection between A and B is established.

Pure p2p networks have the advantage of being completely decentralized; if one node is removed there is not loss of functionality. Utopia behaves in a similar way, if one node is removed from the overlay network, other nodes doesn't loose the functionality of sending messages, since message routing is done with the rest of the neighbors.

Finally the last type of p2p network mention here is the one corresponding to hybrid architectures, in this kind of networks some nodes are router terminal that facilitate the interconnections between peers. Figure 2.15 illustrates more clearly the process when executing a query inside this type of net [16].

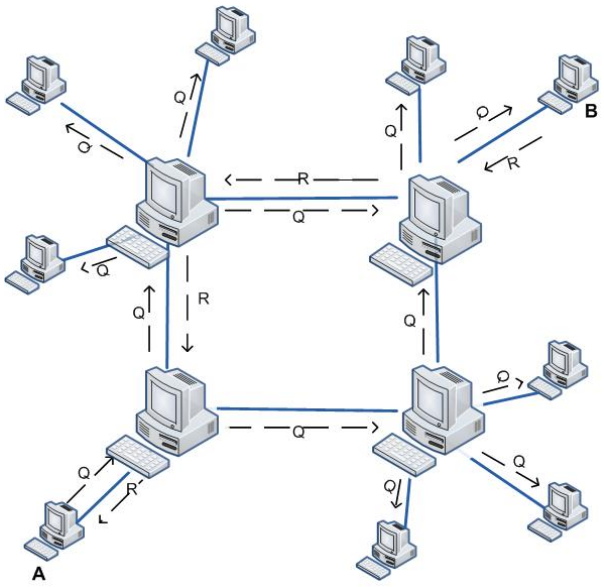


Figure 2.15: Hybrid P2P network architecture

In figure 2.15, node A is trying to find content stored on node B, node A sends query (Q) to his assigned super node (a super node is a node with more resources like bandwidth, speed and computing power that is chosen by the members of the network to perform as such), this super node floods the query similar to the technique used by pure p2p networks, the query Q travels through all the super nodes (therefore faster) until it reaches their target B, B answers through the super node network and finally A knows how to find B.

It is important to mention that if a super node fails or goes down another node is chosen by the rest of the nodes of the network to substitute the place of the fallen super node.

Hybrid p2p networks have the advantage of reducing signaling traffic, since message propagation is done primary through super nodes, consequently saving bandwidth and resources of less powerful nodes. Better search results are achieve thanks to this technique. Sadly, Utopia did not take advantage of such techniques since is not likeable that when choosing a super node it will stay on the same place for a long period of time, physical location of nodes changes fast through time, and even worst maintaining a connection with a node through a long period of time is not likeable to happen.

Figure 2.16 shows in a synthetic way the main advantages and disadvantages of p2p architectures, it also include a comparison with a client - server architecture [22].

<table>
<tr><th>Client and Server</th><th colspan="4">Peer to Peer</th></tr>
<tr><td rowspan="3">Centralize control by the server.

Clients connect only with a server.

Example: www</td><td colspan="4">Decentralize control.

Peers share their resources.

All peers are routers, clients and servers.</td></tr>
<tr><td colspan="3">Unstructured P2P</td><td>Structured P2P</td></tr>
<tr><td>Centralized P2P</td><td>Pure P2P</td><td>Hybrid P2P</td><td>DHT-Based</td></tr>
<tr><td>Central entity informs:

-information available

-which peer has the information

Example: Napster</td><td>No central entity.

All peers have same functionality.

Examples: Gnutella 0.4, Freenet</td><td>Dynamic central entities

SuperNodes: - gives information about network and they are not fixed.

Examples: Gnutella 0.6, JXTA</td><td>No central entity.

Structured fixed.

Table Hash gives:

-Information available and where it is.

Examples: Chord, CAN</td></tr>
</table>

Figure 2.16: P2P's architectures advantages and disadvantages

2.3.3 P2P characteristics

The following section shows how Utopia achieves certain p2p characteristics, and how it takes advantage of them.

Symmetric communication All the devices in a p2p network have the same role and neither is more important than other, following a symmetric communication. In fact a device is server and client and takes one role depending of the recourse requires in each moment. The devices in our application can act as a client when they create and send a message or as a server when it's listening

messages or waiting to forward a message [23].

Decentralized control The control is decentralized, this means that if one node goes down it doesn't affect the overall network; Utopia solves this problem because the control does not depend only of a node. If one node goes down all the other nodes can perfectly keep forwarding message and all system functions are available to the user. In a network which control is not centralizes in a point when the ad-hoc network is created there isn't a single device which controls the rest, therefore decisions concerning a message are taken by all the participating devices. Number and position of participants is unpredictable. Overall network topology change fast all the time since nodes are moving, new devices can join or leave the network at anytime without warning or even knowing that it happened [24].

Self organize P2p has self organize, all the components and their positions are not fixed. Is not necessary to know network topology when sending a message therefore no permanent infrastructure is needed, Utopia can lead automatically with different network topologies without even knowing them. This fact gives more flexibility because the number and positions of devices connected can change without problems or having to refresh the structure of the network. This characteristic facilitates working with mobile phones since they can change of position, leave or join at anytime or simple crash. To have a self organize implies a decentralized administration, there is no single device which administrates the network, every devices has the same role and the same importance. Nodes can act as servers or clients at the same time therefore administration is not centralized and each device can decide by himself where to forward or send the message [24].

Distributed The members of the net don't have a pre-establish position. Since any Bluetooth device running Utopia software can be a member of the network, there isn't a fixed position where they belong, devices are distributed along big areas and their position is unpredictable, some of them change of position faster than others and also have different kind of resources. Members share their resources with others to increase the system efficiency. When forwarding a message each device contributes with their specific available resources to forward a message, resources go from what class of device it is to the simple ability to forward one message [23].

Chapter 3

Design

This section will introduce all the possible scenarios where Utopia works, the uses case of the project and at the end the class diagram of the Utopia project.

3.1 Possible scenarios

In this section, all the possible scenarios which the application has to deal in order to send a message are studied.

Four kinds of scenarios are identified:

1. Direct message to neighbors
2. Send message through one or more intermediaries and get the confirmation message
3. Send message through one or more intermediaries and loose the confirmation message
4. Destination node not reachable

3.1.1 Direct message to neighbors

Destination node is inside of the source´s range. In this case the message is sent directly and we do not need confirmation message because Bluetooth guarantees that the message is received correctly. The Bluetooth API deals with this. This scenario is shown in Figure 3.1.

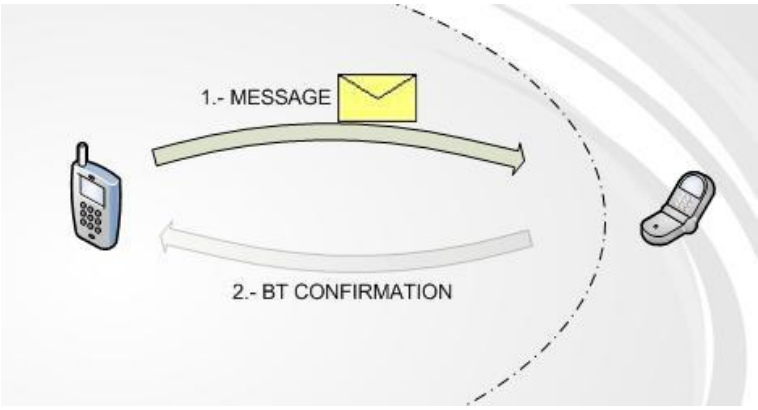


Figure 3.1: Direct message scenario

In Figure 3.1 the source node is the mobile phone on the left of the figure. It sends a message to the mobile phone on the right of the figure and gets back an automatically confirmation message.

If the source node does not receive any Bluetooth confirmation, it means that the source node cannot send the message directly because the destination is not in its range. Figure 3.2 shows this scenario.

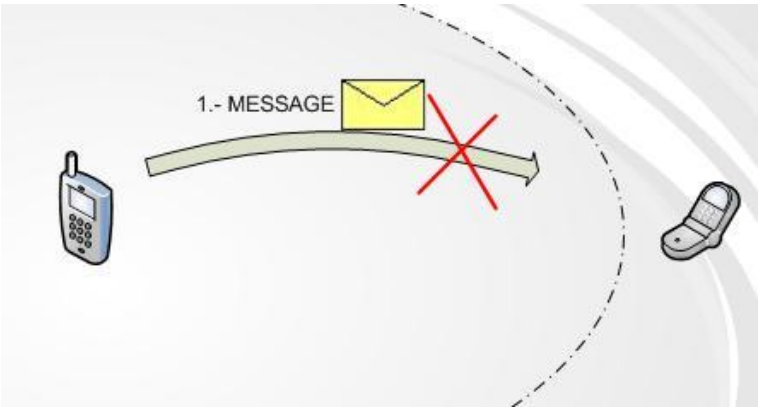


Figure 3.2: Direct message scenario 2

Figure 3.2 shows the scenario when the source node (mobile phone on the left of the figure) try to send a direct message to the destination node (mobile

phone on the right of the figure), in this case the mobile phone is not in its range and the message cannot be received, and the source node will not receive any automatically confirmation message.

In this case the message is going to be delivered through intermediary nodes.

3.1.2 Send message through one or more intermediaries and get the confirmation message

If the client has to use the Utopia algorithm the scenario of Figure 3.3 is presented.

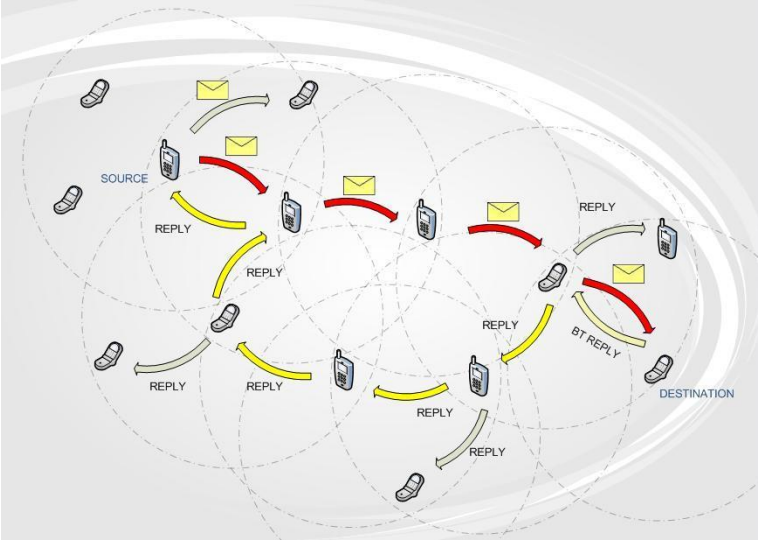


Figure 3.3: Send message through one or more intermediaries and get the confirmation message scenario

In this scenario the source node sends the message to the neighbors using the Utopia algorithm. The message arrives to the destination node through the intermediary nodes.

When the last intermediary node manages to send the message to the destination node, it will get an automatically reply from the destination node and it generates the reply message which has to be sent to the original source node as we can see in Figure 3.4. This is the case when both messages have been received correctly.

3.1.3 Send message through one or more intermediaries and loose the confirmation message

This is the case when the message arrives to the destination node through intermediary nodes and the reply message is lost. As we can see in Figure 3.4 the reason why the source node doesn't receive the reply message is due to the fact that one of the intermediary node moves to another position and no intermediary nodes are able to forward the message. This is because the high dynamic nature of the network.

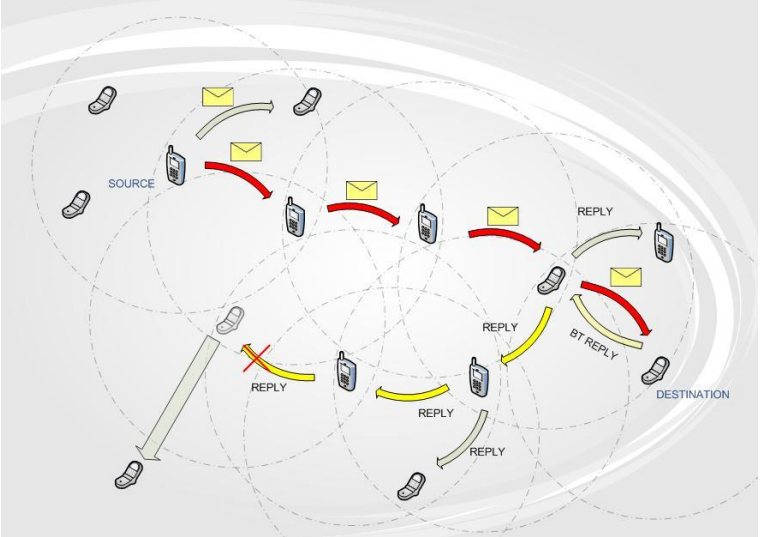


Figure 3.4: Send message through one or more intermediaries and loose the confirmation message scenario

The solution to this problem is to retransmit the message or let the source node to choose if it would like to send a sms instead of a Bluetooth message after retransmitting several times the Bluetooth message. Due to the fact that the source node doesn't know if the destination node received the Bluetooth message or not.

3.1.4 Destination node not reachable

This is the scenario when the destination node is not reachable from the source node even using the intermediary nodes. The solution of this problem is the same of the previous scenario because in both situations the source node doesn't know if the message has been delivered or not.

This scenario is presented in Figure 3.5.

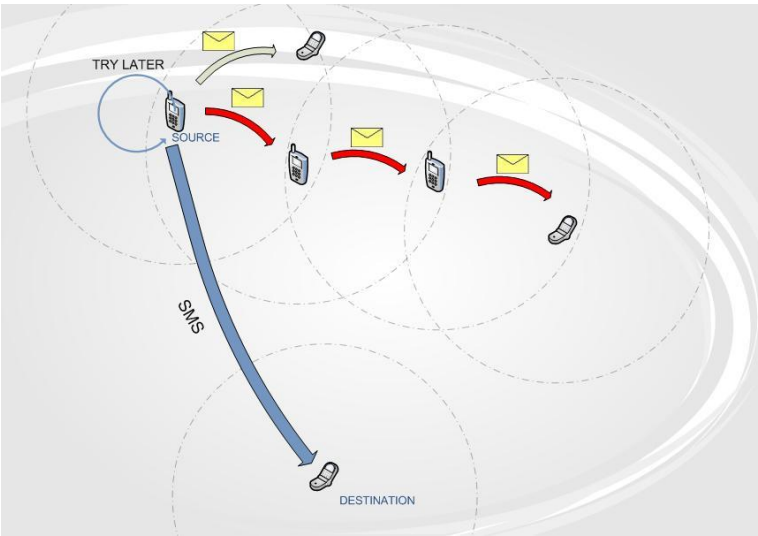


Figure 3.5: Destination node not reachable scenario

In Figure 3.5 the source node has 2 possibilities to deal with this problem, either send a sms or store the message and try to send it later.

3.2 Use cases

Figure 3.6 shows all the possible actions that the user can do using the Utopia application.

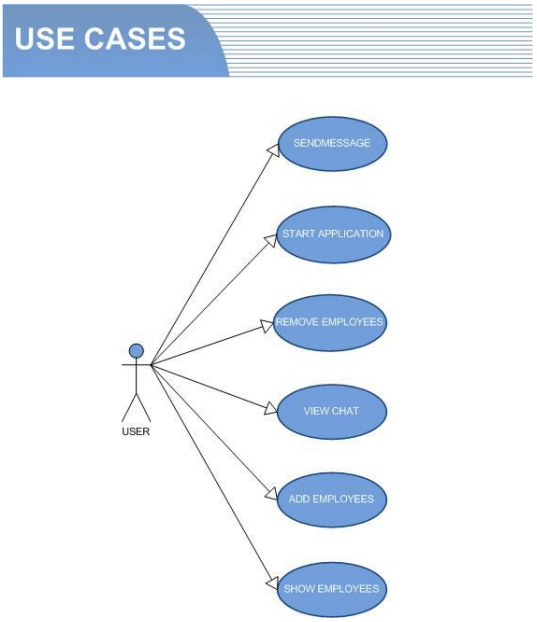


Figure 3.6: Use cases

Figure 3.7 gives a description of the use cases which is shown on Figure 3.6.

USES CASES	DEFINATION
Send message	This action is used when the user would like to send a message to another employee using the Utopia Algorithm after choosing the receiver from the employees list
View chat	This action gives the opportunity to the user show the chat on screen
Add employees	The user uses this action to insert new employees into the database, if new nodes are into the networks or modify the status of an existing node
Start application	The user use the navigation panel to select the SSE1 application, push the enter button and a screen with the main windows is display.
Remove employees	With this action the user can delete from the employees list the devices which is going to leave the network forever.
Show employees	This action gives the opportunity to the user to display on the screen the whole list of employees.

Figure 3.7: Description Use cases

3.3 Class diagram

In this section the class diagram of the project is going to be analyzed. The class diagram presented in Figure 3.8 shows the classes of the system and the operations that each one of them can perform.

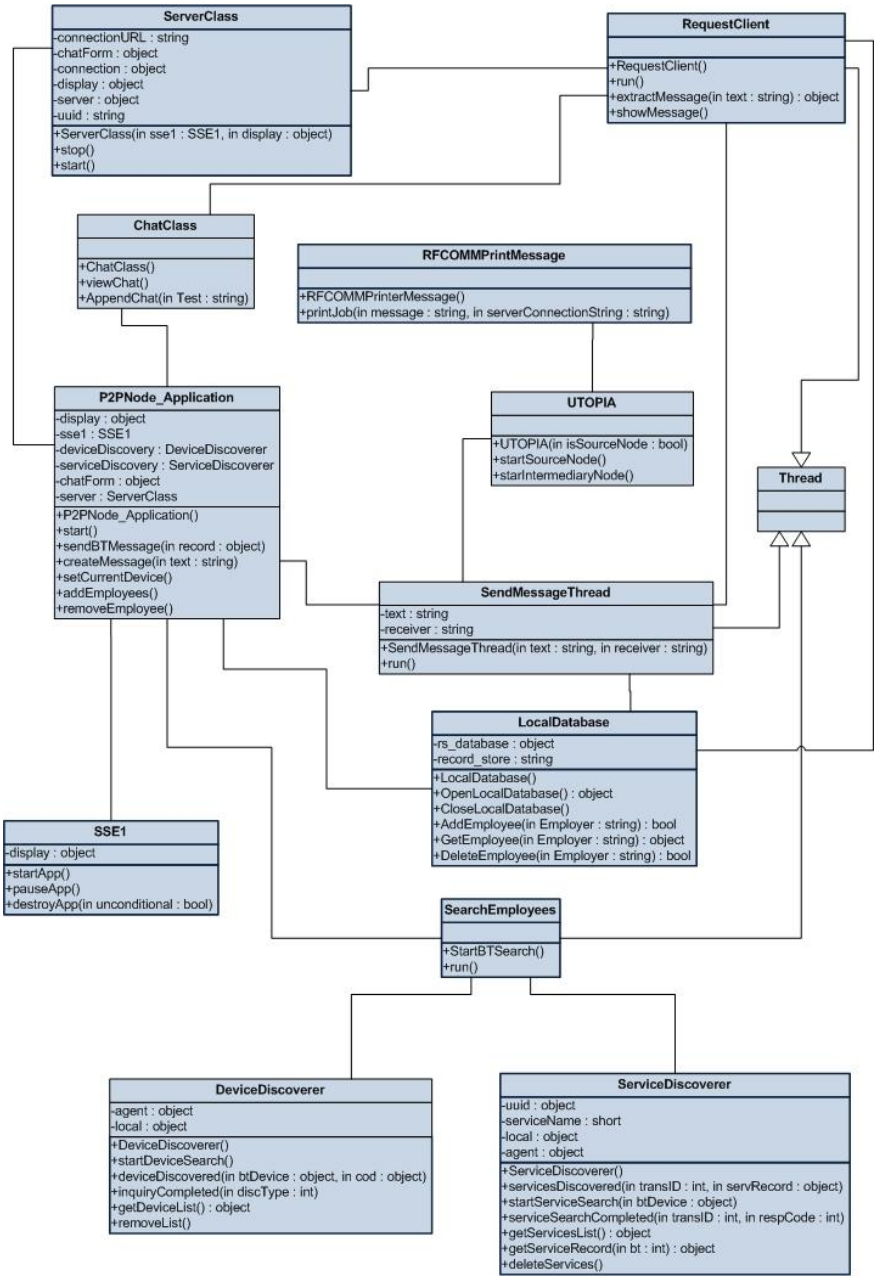


Figure 3.8: Class Diagram

To facility the reader the understanding of this section, some knowledge of the application will be given in advance and the class diagram will be explained later.

When the user executes the application he has to insert some parameters which will be useful for the Utopia algorithm. These parameters are:

- Device class
- Mobility class
- Phone number

More details about these parameters will be given in the Utopia algorithm section. These parameters will be stored in a database which is placed inside the device; such database contains all the employees working in the environment.

The database inserts or updates the employees using the Bluetooth research. Whenever it finds some device it checks if the device is present into the database. If it is inside it updates the current state of the device, else it inserts the state of the device into the database.

When a user wants to send a message to some employee it uses the Utopia algorithm to deliver the message. Whenever a message arrives to the destination node it will be showed on the screen.

The knowledge about how the device uses the Bluetooth research, sends Bluetooth messages, uses the Utopia algorithm and shows the message on the screen will be given in the next section.

As it is shown in Figure 3.8 the class diagram is composed by 11 classes:

1. SSE1
2. P2PNode_Application
3. SearchEmployees
4. DeviceDiscoverer
5. ServiceDiscoverer
6. LocalDatabase
7. SendMessageThread
8. RFCOMMPrintMessage
9. ServerClass
10. RequestClient
11. ChatClass

In this section a closer view of these classes will be given.

The first class is the SSE1 class. It extends the MIDlet class which provides the method to start the application.

After executing the application SSE1 class invokes the P2PNode_Application. P2PNode_Application contains different operations. The method start() displays the list of the employees present on the database. The user can change his current state using the method setCurrentDevice(), this method modifies the mobility of the device, the class of the device and his phone number. The method addEmployees is used to add new Employees to the database. This method invokes the SearchEmployees Class. The method removeEmployee() removes an employee from the database. CreateMessage() method is used to type the message which has to be sent to some employee. The message will be sent using the method sendBTMessage() which has as input the mac address which the message has to be sent.

The SearchEmployees class is invoked to discovery employees and insert or update rows into the database. It uses the DeviceDiscoverer to find Bluetooth devices and the ServiceDiscoverer to investigate if these devices found match with the Utopia requirements. Once a device matches these requirements, it will be stored into the database. This class start the research using the start() method. More details about the Utopia requirements are going to be examined in the next sections.

The DeviceDiscoverer class contains the following methods: startDeviceSearch() to start a device discovery, deviceDiscovered() is invoked when a Bluetooth device is discovered, inquiryCompleted() when the device discovery ends, getDeviceList() returns a vector contains all the Bluetooth devices found and removeList to clean up the vector which contains all the Bluetooth device found.

The ServiceDiscoverer contains the following methods: startServiceSearch() to start the service discovery, serviceDiscovered() is invoked when a device which publishes the service that the process is looking for is found, serviceSearchCompleted() is invoked when the service discovery ends, getServicesList() returns a vector which contains a list of devices which publishes the service that the process is looking for, getServiceRecord() returns a service which the application would like to connects and deleteServices() to clean up the vector contains all the services.

The LocalDatabase class is used to store all the employees and their status into the database. It contains the following methods: openLocalDatabase() is used to access to the database records, closeLocalDatabase() is used to close this resource when it is not needed anymore, addEmployee() is used to add a new employee or to modify an existing employee, deleteEmployee() is invoked to delete a row into the database and getEmployee() is called to get the informations of a specific employee.

When a user would like to send a message to some employee, it invokes the class SendMessageThread which has as input the destination node username

and the text of the message. To deliver the message this class invokes the run method which start the Utopia algorithm.

Utopia class contains the Utopia algorithm, it has as input a Boolean value which specified if the algorithm is invoked by the source node or an intermediary node. If it is invoked by a source node the StartSourceNode method is invoked, otherwise startIntermediaryNode method is invoked. The difference between them is that the StartSourceNode() uses a timer to retransmit the message if the confirmation message is not received. More details about that will be given in the Utopia algorithm section.

The Utopia class delivers the messages using the RFCOMMPrintMessage, this class has only the method printJob() which is in charge to deliver the message to the destination node or the intermediary nodes.

The ServerClass is in charge to running the server and publishes the service SSE1DSProject which is created by the application. It contains two methods: start() which runs the server and stop() which stops the server. When the server is running, it can receive connections from the clients, when it receives a connection it runs the RequestClient thread which deal with it. It contains the following parameters: run() which deal the client requests, extractMessage() which is in charge to extract the received message if it is an intermediary node and showMessage() which invokes the Chat class to append the message on the screen and to show it, if it is a destination node.

The Chat class contains only two methods: appendChat() to write the message on the screen and viewChat() to show the messages on the screen.

In Figure 3.8 it is possible to see that RequestClient, SendMessageThread, SearchEmployees classes extend the Thread class. The reason to use these threads is to manage a concurrently execution in a device. With threads, one device is able to do several actions and synchronize them. More details of using threads will be explained in the architecture chapter.

Chapter 4

Architecture

In this section the architecture of each node of the network is going to be described.

Figure 4.1 shows that in each mobile phone a server and a client are running in the same device. The server publishes a service and client searches services. When a client starts a device discovery, for each one discovered, it has to start also a service discovery. This way it shows only the devices which publishes the researched services, and discards the others. When a client finds the desired service running in a server, it can establish a connection with that server.

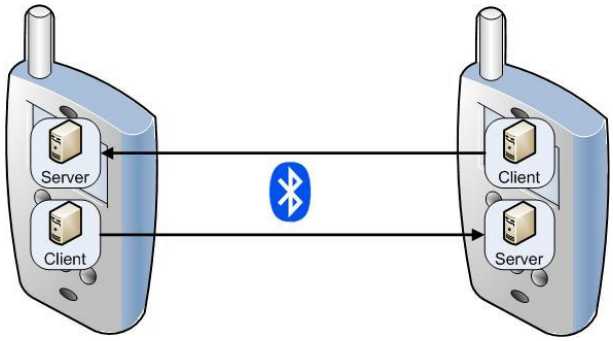


Figure 4.1: Architecture

Following a description of the server and the client is given.

4.1 Server

A really important step that a server performs is to register a service, due to the fact that when a Bluetooth device connects to another device, what we are doing is accessing a service offered by the other device.

In Figure 4.2 the server's steps are shown.

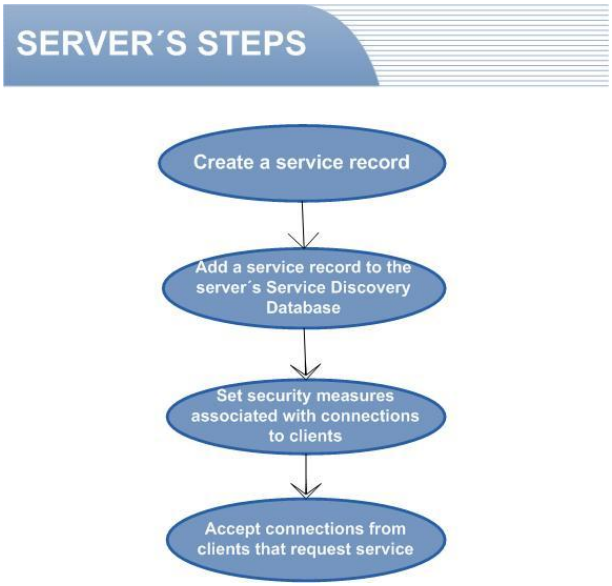


Figure 4.2: Server's steps

The first step is to create a service record. A service record is sufficient information to allow clients to connect to the Bluetooth service being offered by the server. A service record contains a list of UUIDs which represent the type of service being offered and every service has a UUID. UIDDs (Universally Unique Identifiers) are 128 bit values which identify a unique Bluetooth service [25].

After that we have to add the services record created to the Service Discovery Database. The Service Discovery Database keeps service records corresponding to the services offered by the devices.

In the third step we have to set the security measures associated with connections to clients. The possible security measures are:

- Authentication
- Authorization
- Encryption

The first security measure is based on a PIN number shared between devices. It refers to the process of verifying the identity of a remote device. The Bluetooth authorization is the process by which a server device grants a specific client access to a specific service it offers. The last security measure is used in a Bluetooth communication to protect sensitive data from eavesdropping. It requires the previous authentication of remote device. All of these settings should

be added as parameters to the connection URL [25].

After setting these parameters the server is ready to run and receive connections from the clients.

In Figure 4.3 the code of Utopia project server is shown.

```
13  public class ServerClass
14  {
15      private LocalDevice local;
16      private StreamConnectionNotifier server = null;
17      private byte[] data ;
18      private int length;
19      private boolean running = true;
20      private static final UUID uuid=new UUID("0a6aa7a65e86465cacf79ee9a3fbf654",false);
21      private StreamConnection conn = null;
22      private static String connectionURL = "btspp://localhost:"+uuid.toString()+
23              ";authenticate=true;authorize=false;encrypt=true;name=SSE1DSProject";
24      private Display display;
25
26      public ServerClass(Display display)
27      {
28          this.display=display;
29      }
30
31      private void runServer()
32      {
33          running = true;
34          StreamConnection connection;
35          try
36          {
37              local = LocalDevice.getLocalDevice();
38              local.setDiscoverable(DiscoveryAgent.GIAC);
39              server = (StreamConnectionNotifier)Connector.open(connectionURL);
40              ServiceRecord record = local.getRecord(server);
41              record.setDeviceServiceClasses(0x40000);
42          }
```

Figure 4.3: ServerClass class code

In Figure 4.3, in the lines 21 and 22, the UUID number of 32 bit is defined. In the next two lines URL connection is composed of UUID number and the following parameters:

- Authenticate = true
- Authorization = false
- Encrypt = true

Due to the fact that Utopia project will be used on restrict area as university or any kind of job environment, we have set the Authenticate and Encrypt to true. In this way a pin code will be asked only at the first time that a connection is established. Afterwards if two devices try to connect again, the pin code will not be asked anymore because these two devices are in paired state.

Authorization is set to false otherwise for all messages received from the server, the users have to insert the same pin code of the client. Utopia's project goal is to send messages to devices which are not in the range of the sender using other devices, if each of the intermediaries requires the same pin code; we

cannot work in background with them.

Regarding the security it is important to underline that the server can receive messages exclusively from devices which are running the same application.

In figure 4.3, from the line 31 to 42 the runServer method is defined. When the class ServerClass is initialized a thread will run, the server will publish a specific service called *SSE1DSProject* which contains the parameters above (lines 42-46).

Figure 4.4 shows server behavior code when it is running.

```
57          while (running)
58          {
59              try
60              {
61                  conn = server.acceptAndOpen();
62                  new RequestClient().start();
63              }
64              catch (IOException e)
65              {
66                  System.err.println("IOException: " + e.getMessage());
67                  return;
68              }
69          }
70          if (conn != null)
71          {
72              try
73              {
74                  conn.close();
75              }
76              catch (IOException e)
77              {
78              }
79          }
80          try
81          {
82              server.close();
83          }
84          catch (IOException e)
85          {
86              System.out.println("Failed to start service\nIOException: " + e.getMessage());
87          }
88      }
```

Figure 4.4: ServerClass class code 2

Figure 4.4 shows that the server is ready to receive connection from some client. The *acceptAndOpen* call (line 61) blocks the server till a client connects to it. When a connection is established a new thread is created which is in charge to handle client's requests.

In Figure 4.5, the behavior of the server is shown.

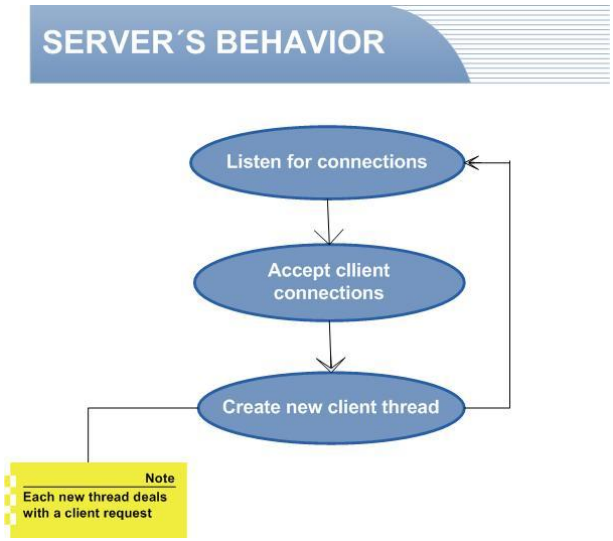


Figure 4.5: Server's behavior

When the server accepts a connection it will run a new thread which deals with the requests of the client and the old thread is continuing to run waiting for other client's requests. In this way the server will not accept the connections sequentially but it can serve them simultaneously through the use of threads - one thread per each client connection.

Figure 4.6 shows the possible tasks that the server has to perform when it receives a message from other devices. It is handled on the RequestClient Thread.

```
126     class RequestClient extends Thread
127     {
128         public RequestClient()
129         {
130
131         }
            public void run()
133         {
134             DataInputStream in = null;
135             try
136             {
137                 in = conn.openDataInputStream();
138                 length = in.readInt();
139                 data = new byte[length];
140                 in.readFully(data);
141                 final String stringa = new String(data, 0, length);
142
143                 if(stringa.equalsIgnoreCase("RefreshTheNeighborsTable"))
144                 {
145                     // Don't do anything
146                 }
147
148                 else if(stringa.equalsIgnoreCase("IntermediaryNode"))
149                 {
150                     // Start Eutopia Alghrotm
151                 }
152                 else
153                 {
154                     // Show the message
155                 }
```

Figure 4.6: Client's requests

When the server receives a client's request, it has to open a DataInputStream to get the request. In the line 137 a DataInputStream is created from the connection that the client had already established (line137). At beginning it extracts the length of the message (line 138), creates a byte array of that lenght to store the message (line139). After getting the message the byte array is converted in a String object (line 141).

Once that the server received the message it has to interpret the client's request . The client can send 3 different requests:

1. The client asks the server if it is alive (line 143), in this case the server will not do anything, because if the client manage to connect to the server service, it will get an automatically confirmation message (lines 143 - 146). This operation is done to make a fast device discovery.

2. The client asks the server to find a specific user. In this case the server is in an intermediary position of the network and it has to start the Utopia algorithm (lines 148 - 151).

3. The server is the destination node. In this case it will just show on the screen the received message (line 153 - 155).

More details of how the server will interpret the request, and how it reacts to these requests is going to be given on the next chapter.

4.2 Client

In the server part, it is explained that when a server is running, it publishes services. What the client does is to locate and to access to these services.

In Figure 4.7, it is shown the steps that the client does to locate a service.

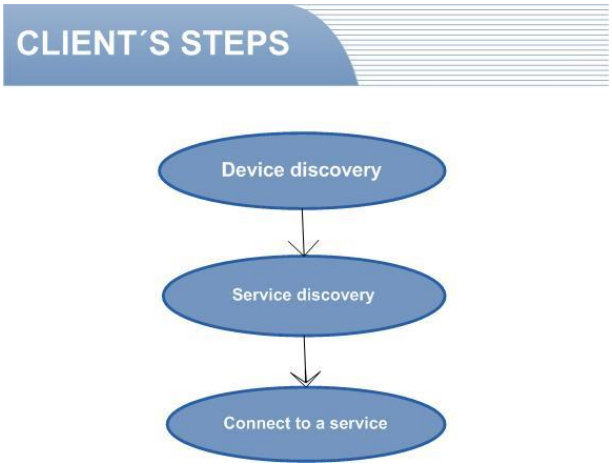


Figure 4.7: Client's steps

4.2.1 Device discovery

The first step that the client does is the device discovery. The device discovery will search all possible Bluetooth devices. To do this we have implemented a *DiscoveryListener*.

In Figure 4.8, it is explained the code of our DiscoveryListener that Utopia project uses.

```
 8  public class DeviceDiscoverer implements DiscoveryListener
 9  {
10
11      private DiscoveryAgent agent;
12      private Vector remoteDevices;
13      private RemoteDevice [] devices = null;
14      private LocalDevice local;
15      private boolean access;
16
17      public DeviceDiscoverer() throws BluetoothStateException
18      {
19
20          remoteDevices  = new Vector();
21          local = LocalDevice.getLocalDevice();
22          agent = local.getDiscoveryAgent();
23
24      }
25
26      public void startDeviceSearch()
27      {
28          access=false;
29          try
30          {
31              agent.startInquiry(DiscoveryAgent.GIAC,this);
32          }
33          catch (BluetoothStateException ex)
34          {
35              ex.printStackTrace();
36          }
37      }
```

Figure 4.8: Discovery Listener

As the first step we need to obtain a DeviceDiscoveryAgent (lines 21-22). We start the research calling the startInquiry method. This method takes in input a DiscoveryListener and an access code which can be:

- DiscoverAgent.GIAC
- DiscoveryAgent.LIAC

The first one indicates an unlimited search returning all devices found in the Bluetooth range. The second one return only remote devices [25]. Utopia uses DiscoveryAgent.GIAC because all the devices which are in the Bluetooth range has to be found.

Each time that the research finds a Bluetooth device the method deviceDiscovered is called.

Figure 4.9 shows the deviceDiscovered method.

```
40  public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod)
41  {
42      if((cod.getMajorDeviceClass()==0x200))
43      {
44          if(( cod.getServiceClasses() & 0x40000) != 0)
45          {
46
47              remoteDevices.addElement(btDevice);
48          }
49      }
    }
```

Figure 4.9: Device discovered method

Doing the device discovery all the Bluetooth devices are returned, probably there are many of them that are not running the Utopia application, that's why some filters are implemented to get only devices which satisfy certain conditions.

Lets suppose that we are interesting only on mobile phones, the implementation should accepts only Bluetooth devices which are mobile phones and discards the others.

The first filter (major devices classes) that we use is implemented in the line 41: the Bluetooth devices found should be a phone. In this way when it will found computer, pda, printer it will discard them. A device which satisfies this condition is added to the *remoteDevices* vector (line 46).

In Figure 4.10, it is shown some example of major devices classes.

Major Device Class	*Example*	*Hex Value*
Computer	Desktop, laptop, PDA	0x100
Phone	Cellular,cordless,modem	0x200
LANAP		0x300
Audio/video	Headset,speaker,video display, VCR	0x400
Peripheral	Mouse,joustick,keyboard	0x500
Imaging	Printer, scanner, camera, display	0x600

Figure 4.10: Major device classes

When the inquiry is completed the method inquiryCompleted is called. This method returns different states of the termination [25]:

- INQUIRY_COMPLETED: Which indicates that the inquiry has been terminated in a normal way
- INQUIRY _TERMINATED: Which indicates that the inquiry has been terminated by the application using the cancelInquiry method of *DiscoveryAgent*
- INQUIRY_ERROR: Which indicates that the inquiry failed and it was not terminated

If the inquiry is completed successfully, the vector remoteDevices, which contains our filtered devices, is returned for further processing.

Figure 4.11 shows the behaviour of the device discovery.

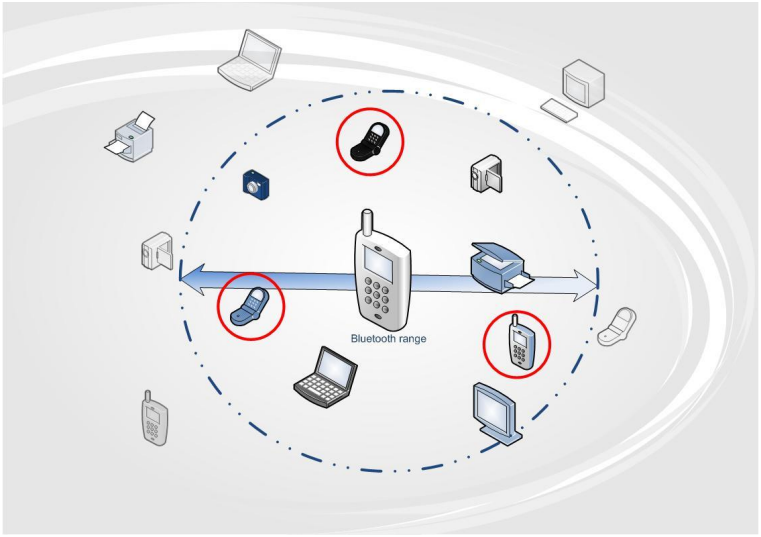


Figure 4.11: Discovery device behaviour

In Figure 4.11 the device in the centre is the one which is looking for Bluetooth devices, the mobile phones marked in red are filtered from the deviceDiscovery method. All the other devices are discarded because they are not in the set 0x200 which is the major class phone.

4.2.2 Service Discovery

Once the device discovery ends the filtered devices are given to the service discovery which searches the required service. This mechanism is similar to the device discovery. The first step is to implement a service search DiscoveryListener.

Figure 4.12 shows the *ServiceDiscoverer* class implementation.

```
3  public class ServiceDiscoverer implements DiscoveryListener
4  {
5      private LocalDevice localDevice;
6      private DiscoveryAgent agent;
7      private UUID uuid = new UUID("0a6aa7a65e86465cacf79ee9a3fbf654", false);
8      private UUID[] uuidSet = {uuid};
9      private static final String SERVICE_NAME = "SSE1DSProject";
10     private static final int[] attrSet = {0x200};
11     private Vector services = new Vector();
12
13     public ServiceDiscoverer()
14     {
15         try
16         {
17             localDevice = LocalDevice.getLocalDevice();
18
19         }
20         catch (BluetoothStateException ex)
21         {
22             ex.printStackTrace();
23         }
24         agent = localDevice.getDiscoveryAgent();
25     }
```

Figure 4.12: Service Discovery Listener

First, a *DiscoveryAgent* is obtained and it will start a service search (line 24). On the line 7 a representation of the *UUID* is created and included into the *uuidSet*. The set *uuidSet* contains all the uuids which the service discovery looking for. In Utopoia implementation, this set in composed by one uuid.

The second step is to call the following method for each remote device found with the *deviceDiscovery* method [25]:

agent.searchServices(attrSet,uuidSet,btDevice,this);

- *attrSet* is an array contains the attributes that we want to retrieve from the *ServiceRecord* of the service found
- *uuidSet* is an array which contains all the uuids number that we are searching
- *btDevice* is the remote device which we are inspection
- *this* is the listener that we are using to inspect the remote device

In figure 4.13 the *startServiceSearch* method and the *serviceSearchCompleted* methods are shown.

```
68      public void startServiceSearch(RemoteDevice btDevice)
69      {
70          try{
71              agent.searchServices(attrSet,uuidSet,btDevice,this);
72          }catch (BluetoothStateException ex){
73              ex.printStackTrace();}
74          synchronized (this)
75          {
76              try{
77                  this.wait();
78              }catch (Exception e){
79                  System.out.println(e);}
80          }
81      }
        public void serviceSearchCompleted(int transID, int respCode)
83      {
84          String message;
85          if(respCode == DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE)
86              message = "Device not reachable";
87          else if(respCode == DiscoveryListener.SERVICE_SEARCH_NO_RECORDS)
88              message = "Service not available";
89          else if(respCode == DiscoveryListener.SERVICE_SEARCH_COMPLETED)
90              message = "Service search completed";
91          else if(respCode==DiscoveryListener.SERVICE_SEARCH_TERMINATED)
92              message = "Service search terminated";
93          else if (respCode==DiscoveryListener.SERVICE_SEARCH_ERROR)
94              message = "Service search error";
95
96          synchronized (this)
97          {
98
99              this.notifyAll();
100         }
101     }
```

Figure 4.13: startServiceSearch and serviceSearchCompleted methods

Each time that a remote device has to be inspected the *startServiceSearch* method is called. In the line 71 of Figure 4.13 the searchServiceSearch is invoked, and from the line 74 to 80 it is determined if another search can be started. If not, waits for a service search to end.

When the search is completed the *serviceSearchCompleted* is invoked, and if a thread is waiting it is notified so another search service process can start (line 96-100). This method returns a response code and thanks to this code it is possible to know the success of the search [25]:

- SERVICE_SEARCH_COMPLETED: Which indicates that the search has been terminated in a normal way
- SERVICE_SEARCH _TERMINATED: Which indicates that the inquiry has been terminated by the application using the *cancelServiceSearch* method of *DiscoveryAgent*
- SERVICE_SEARCH_ERROR: Which indicates that the service search has been terminated with an error
- SERVICE_SEARCH_NO_RECORDS: Which indicates that the remote device has not service records

- SERVICE_SEARCH_NO_REACHABLE: Which indicates that the remote device is not in the range anymore

If the required service is found, the method *serviceDiscovered* will be invoked. Figure 4.14 shows the serviceDiscovered method.

```
53  public void servicesDiscovered(int transID, ServiceRecord[] servRecord)
54  {
55      for(int i=0;i<servRecord.length;i++)
56      {
57          DataElement serviceNameElement = servRecord[i].getAttributeValue(0x200);
58          String serviceName = (String)serviceNameElement.getValue();
59          if(serviceName.equalsIgnoreCase(SERVICE_NAME))
60          {
61              ServiceRecord serviceRecord = servRecord[i];
62              services.addElement(serviceRecord);
63          }
64          else
65          {
66              services.addElement(null);
67          }
68      }
```

Figure 4.14: Service Discovered method

In *serviceDiscovered* method investigate if the service that is found has the same name of the service that it is looking for (lines 58-62). If they have the same name this element is added to a vector services which is used for further processing. Otherwise this element is set to null and it is added to the same vector(line 65).

Figure 4.15 shows the behaviour of the service discovery.

Figure 4.15: Service Discovery behaviour

In Figure 4.15 the service discovery is investigating each mobile phone fil-

tered from the device discovery. The mobile phone marked in red is not able to connect with the phone in the centre of the picture because his server is not publishing the service called *SSE1DSProject*. The other two mobile phones are inserted on his list because their server is publishing the service which the mobile in the centre is looking for.

4.2.3 Connect to a service

Once a device which satisfies all these requirements is found, a connection could be established. In order to connect with another device a getConnectionURL method is used.

public String getConnectionURL(int requiredSecurity, Boolean mustBeMaster)

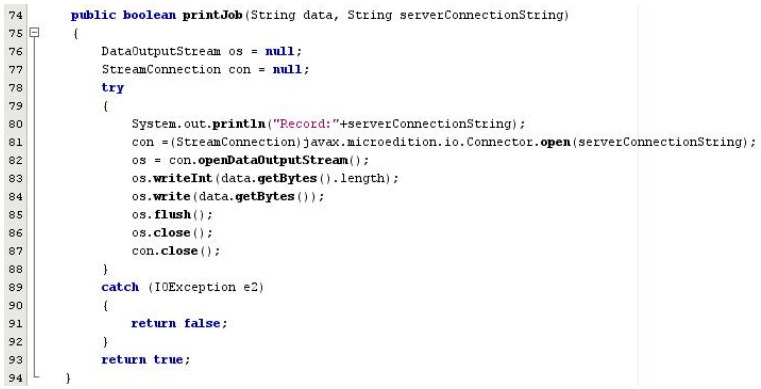
This method allows connecting with the service. The first input parameter indicates the level of security for the connection.

The second argument allows being master or slaving in the connection. Master/Slave is a model for communication protocol in which one process (master) controls another processes (slave). Once a master and a slave established a relationship, the master is always in charge of the control. If we set this Boolean value true, the device plays the role of master; false indicates that the device could be either the master or the slave.

Due the fact of Utopia has been tested on mobile phones running OS Symbian, mustBeMaster has to be set to false because the current implementation of Bluetooth API on Symbian OS supports only a value of false for the *mustBeMaster* parameter. If mustBeMaster is set to true an exception is received [25].

After getting the URL connection using the method *getConnectionURL* Connection could be opened.

Figure 4.16 shows how the Utopia application manages to connect to the right service and sends messages.



```
74      public boolean printJob(String data, String serverConnectionString)
75      {
76          DataOutputStream os = null;
77          StreamConnection con = null;
78          try
79          {
80              System.out.println("Record:"+serverConnectionString);
81              con =(StreamConnection)javax.microedition.io.Connector.open(serverConnectionString);
82              os = con.openDataOutputStream();
83              os.writeInt(data.getBytes().length);
84              os.write(data.getBytes());
85              os.flush();
86              os.close();
87              con.close();
88          }
89          catch (IOException e2)
90          {
91              return false;
92          }
93          return true;
94      }
```

Figure 4.16: Print job method

The method printJob is in charge of connecting to the right service (serverConnectionString - line 81) which is given as an input parameter. At beginning a DataOutputStream stream and a StreamConnection connection are initialized. In the line 81 the connection is established. In the line 83 the length of the message is written on the output stream, next the bytes of the message are written in the output stream and in the line 85 it flushes this data output stream.

When all these operations end the resources have to be closed. In fact, in the lines 86 and 87 the DataInputStream and the StreamConnection are closed.

Chapter 5

Utopia algorithm

In this section the UTOPIA algorithm is explained. Utopia algorithm is used when sending messages to nodes which are not inside the Bluetooth range.

There are three different types of nodes, source, intermediaries and destination; each one plays important roles inside Utopias algorithm.

An intelligent structure of a database is needed in order to save time and memory since mobile phones works with limited storage resources; such structure is closely related with a Neighbor table which is used to be aware of nodes within the range of a source node.

Size of networks plays an important role inside any ad-hoc network, since performance and efficiency are directly connected with it.

Different sets of rules and norms represent the heart of Utopias engine, two important roles are part of such rules; being a source or initiating node and the opposite just a simple forwarding node.

5.1 Type of the node

Nodes can play different roles depending of what they do with a message. A message can be sent, forwarded or received to or from some other node, therefore devices can play: source, intermediary or destination roles .

In this section a closer look to different the kind of nodes is going to be given.

A **source node** sends messages to some destination node inside the network. It creates messages and starts the Utopia algorithm.

The task of the **intermediary node** is forwarding messages. Before forwarding a message it checks if the message should be destroyed, in order to avoid loops, if the message doesn't find the destination node.

Another important action is to generate the confirmation messages. It is important to generate it, because the source node needs to ensure that the message has been received. This action belong to the last intermediary node, which find the destination node. It sends the message to the destination node and receives from it a Bluetooth confirmation message.

Bluetooth uses *ARQ (Automatic Retransmission Request)* controls lost packages in a transmission. ARQ uses an acknowledgment (ACK) which, if the sender does not receive an ACK it re-transmits until receive an ACK or it expired the number of re-transmit [27]. Intermediary node forwards confirmation messages too.

A **destination node** receives the message and shows it to the user.

Figure 5.1 shows these three different kind of nodes.

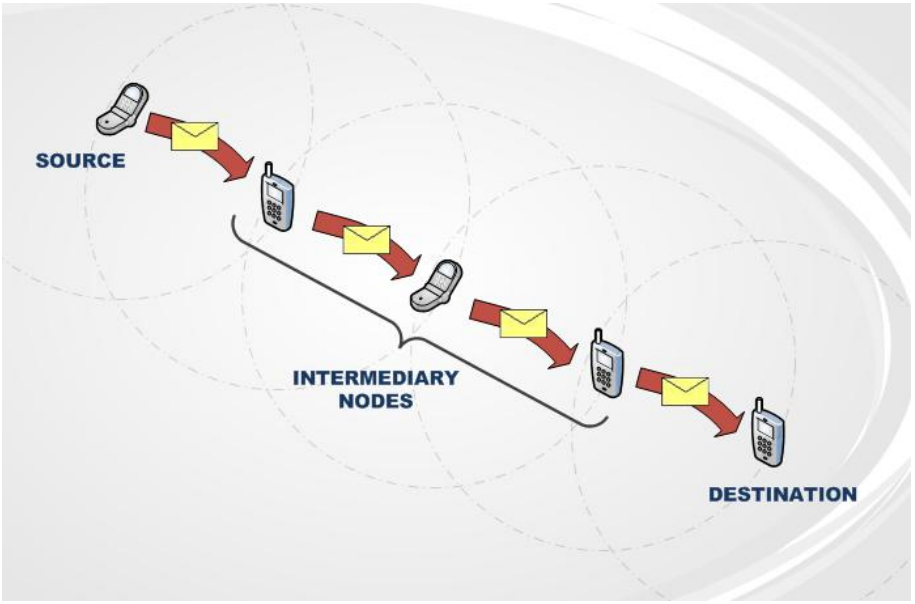


Figure 5.1: Different kind of nodes

5.2 Structure of the database

Utopia main repository of information is a database located inside the internal memory of devices.

Figure 5.2 shows an example of the structure of the internal memory of each device.

Username	MAC Address	Class	Type	Phone Number
Johnbus	E2-3F-5A-3D-5A-4B	1	printer	03888892
Mikiplan	1B-5C-8A-1B-5E-2E	2	laptop	45667521
Meganecof	4D-5B-5E-6A-7B-1D	2	mobile phone	48943011
Cletusmark	5A-EE-4D-5D-CB-9C	1	computer	78532109
Suzefar	5A-EA-56-54-1D-3C	3	mobile phone	95400034
Willdev	6B-E3-F6-5D-6F-77	2	mobile phone	34386879
Simsed	43-3B-CC-67-BA-8A	2	mobile phone	56555543
Molamazo	56-4F-3D-1D-1B-4F	1	printer	46387486

Figure 5.2: Mombile phone database

The **username** is the Bluetooth name of the device. In Utopia this username is unique due to the fact that it is used to access to the internal memory.

The **mac address** is the unique address of each Bluetooth device. Thanks to this address devices are able to communicate.

The **class device** is really an important parameter because it sets the range of the Bluetooth devices.

The last parameter is the mobile **phone number** which gives the opportunity to the algorithm to deliver the message using a normal sms, if the destination node is not available in the network.

5.3 Neighbors table

The neighbors table is a really important part of Utopia algorithm. It includes only the devices which are in the Bluetooth range.

The Bluetooth API to generate the neighbor table uses the device discovery and the service discovery. During testing the Utopia application, it figures out that this process is really slow, so doesn't match with a high dynamic networks.

For this reason a new device discovery faster than the real device discovery is implemented called Utopia fast device discovery (UFDD).

Utopia project uses the device discovery and the service discovery when it has to add a new node in the network, because it has to store the properties of the new node in the internal memory. This is the only case when Utopia uses the device and service discovery, otherwise it uses the UFDD.

In order to create the neighbors table UFDD is used. The algorithm obtain form the internal memory all the mac addresses present in the database and it checks if they are in the range Bluetooth. To understand if a node is in its range Bluetooth it sends a message like this: "Are you alive?", when the server of the other node receives this message, it doesn't react because the API Bluetooth guarantee if the message has been delivered. If the message had delivered the username of the node which received the message will be inserted in the neighbors table.

5.4 Size of network

The size of the network which our application uses is fixed. This is assumed because nodes need to know exactly which devices are able to receive messages. The way to manage this is to have a complete list with all the employees that belong to the network; the database explained in previous section is in charge of this.

The size of the network can change in the following way:

- A node is removed a node from the network
- New node is added in the network

The first case is when a employee is fired out, the row with his information is removed from the internal memory of each device. The second case is when a new employee is hired and all his information needs to be inserted into the database.

5.5 Structure of the message

In this section the important concepts of the messages are going to be studied. At beginning the structure of the message which is going to be sent will be defined, than the forward messages sturcture will be shown, and the reply messages will be analyzed. At the end the cycle life of the message will be showed.

Two different kind of messages are presented in Utopia algorithm:

1. normal message (RS)
2. replay message (RSReplay)

A normal message (RS) is the message that source node would like to send to a destination node, and a replay message (RSReplay) is the confirmation message created by the last intermediary node which indicate that the message has delivered.

Figure 5.3 shows the structure of the normal message (RS).

Figure 5.3: RS structure - forward message

As it is shown in Figure 5.3 the forward message structure is composed by:

- **Predecessor** is the last intermediary node which forwarded the message
- **N_iteration** is an integer number which is increased when the source node retransmit the message because it doesn't receive the confirmation message. This number represents the current Utopia level. It means that the intermediary node which receives the message that has to be forwarded knows the current level of Utopia algorithm. The max number of N_Iteration is equal to the size of the network because in this way the algorithm will flood the networks
- **ID** is an identification number, which is increasing when the node sends a message to some other node
- **Source** field is the Bluetooth name of the source node which sends the message to the destination node
- **ID** and **Source** is the primary key of the message. Thanks to these two fields it is possible to identify the message
- **Destination** filed is the destination node which should receive the message
- **N_hopes** is the number of hops that a message can be forward through a specific number of intermediary nodes. This number is equal to the number of the size network and it decreases when the message is forwarded. It is a really important field due to the fact that it stops message propagation to infinity.
- **Message** field is the text of the message that the source node would like to send to the destination node

If the source or intermediary node will send directly the message to the destination node the structure of the message is reduced as it is shown in figure 5.4.

Figure 5.4: RS structure - direct message

The destination node will recevice only these fields:

- **ID** and **Source** which identify the message. Thanks to these fields, if another intermediary node delivers the same message, the destination node is able to recognize the message and shows it on the screen only one time.
- **Message** which is the message that will be showed on the screen

Figure 5.5 shows the confirmation message structure. This message is created by the last intermediary node which gets an automatic acknowledgement from the destination node. Confirmation message is sent to the source node (which became the destination node of the confirmation message) by actual destination node. This structure of the reply message is used when the reply message is forwarded to some nodes which are not the destination node.

Figure 5.5: RSReply structure - forward reply

As it is shown in Figure 5.5, RSReplay includes the following fields:

- **Predecessor** which is the last intermediary node that sended the reply message
- **Id** and **Source** which identifies the message sent by the source node
- **N_hopes** which has the same behaviour of the RS N_hopes

If the destination or intermediary node sends directly the reply message to the source node, the strucuture of the reply message is shown in figure 5.6.

Figure 5.6: RSReply structure - direct reply

As it is shown in the figure 5.6, this reply message is composed by **ID** and **Source**. So if the source node receives more than one reply message, it will shows the corresponding alert message on the screen only one time.

5.5.1 Two way handshakes

Utopia uses a 2 way handshake protocol, therefore reply messages are sent using Utopias algorithm with the maxmium level of iterations which is euqal to use a broadcast or flooding technique. Reply message is broadcasted in order to increase the chances of delivering it, since rentransmissions are not done for this kind of message and destination node never recives an acknowledgement for it.

Figure 5.7 shows the two ways handshakes.

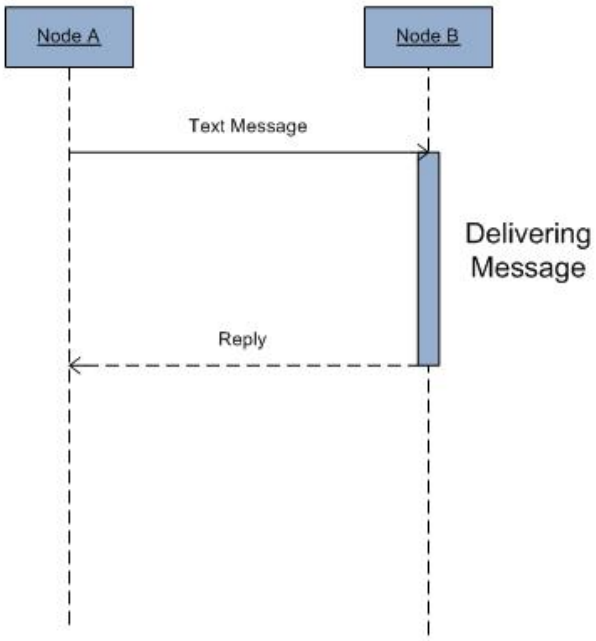


Figure 5.7: Two way handshakes - scenario A

Figure 5.7 shows an example of two ways handshakes between two nodes. The node A sends the message to node B and it receives a reply message to be ensure that the message was delivered.

Figure 5.8 shows an example where a message is sent between source and destination nodes. Intermediary node receives an automatic acknowledgement from the destination and sends a RSReplay message to the source node using broadcast techniques

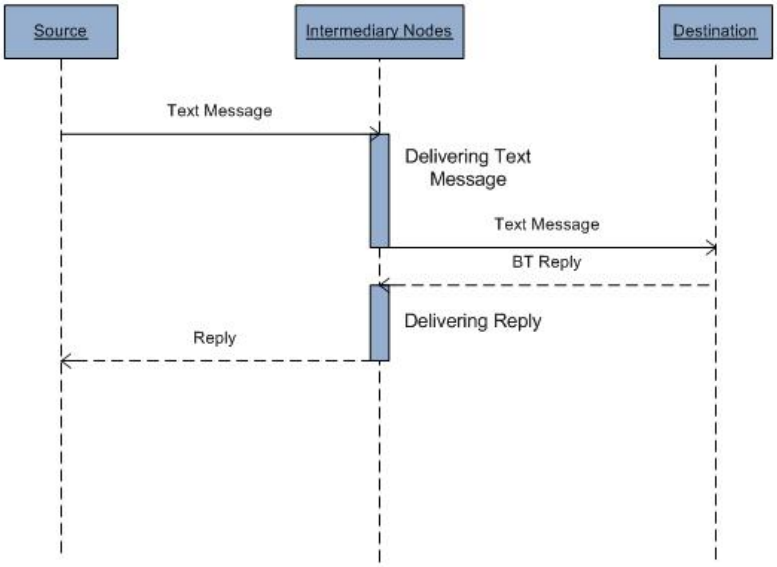


Figure 5.8: Two way handshakes - scenario B

5.5.2 Message state transition diagram

Due to the importance of the message management, this section explains the different states of the message life cycle.

Figure 5.9 shows the life-cycle of the message.

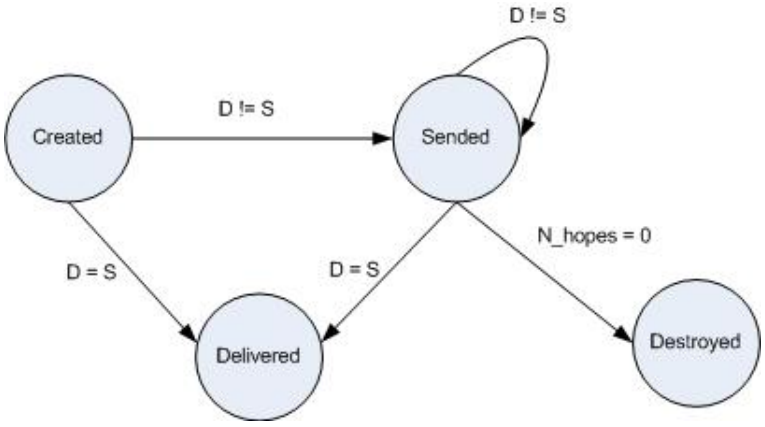


Figure 5.9: Message life cycle

A message can have different states while it is forwarded. When the source node creates the message, it is in the initial state *Created*. As explained before the message can be delivered in two ways. The first way is when the destination node is a source's successors: in this case the message state goes from the *Created* state to the *Delivered* state. In the other case the destination node is not a source's successor, the message state changes from *Created* to *Sended*. *Sended* state indicates that the message is forwarded to other intermediaries nodes until one of them has as neighbour the destination node, if so it sends the message to the destination and the message state changes from *Sended* to *Delivered*.

While the message is being sent the node has to check the number of hops and when it reaches 0 the message is destroyed to avoid message propagation to infinity.

5.6 Protocol

This section explains different protocols executed by Utopias algorithm, each one of them is reponsable for an specific task. The following topics are cover:

- Source node protocol
- Forward node protocol
- Waiting timer
- Number neighbors
- Neighbor selection protocol

5.6.1 Source node protocol

The starting point of all Utopia procedures is when a message is created and sent for the very first time, important sets of rules are run when such situation happens, the responsible of this operations is the source node.

The protocols run inside source node assure that the network is flooded if everything else fails by keeping track of message iteration, more specifically the waiting timer protocol is responsible for this, such protocol runs inside source node protocol and it's an important part of it.

Utopias broadcast retransmission effort stops after a random period of retries between 1 and 3, such number can be adjusted to fit different environments where message correct delivery is more or less important, if it is more important a higher random number can be used similar to adjust N_Hops initial and expansion value which represent the depth of each iteration. Choosing this kind of settings it's a game play between less overhead and better rate of message delivery.

You can think on random period of retries as the number of explosive waves you can see after a nuclear explosion. Each wave represents a new big effort (by completely flooding the network) of trying to find the destination node. It is true that Utopia algorithm generates way much more overhead than a normal flood technique, this is because the high dynamic nature of the network. Each time a message is broadcasted, network topology may be in a completely different state than it was before.

Since message delivery is crucial, some performance and efficiency issues (like more flooding and overhead) are sacrificed in order to achieve better message delivery rate. Utopia leaves open a lot of special settings to adjust its behavior to different kinds of scenarios and environments, example of those settings are n_Hops initial and expansion rate, initial iteration and iteration expansion rate, number of retransmissions and initial timer values.

Figure 5.10 shows pseudo code corresponding to this protocol. Important lines are marked in red and commented below.

```
2  sendMessage(message)
3
4    If the destination (D) is your neighbor
5        Send message directly
6           Send(D,message)
7
8
9     Else
10       Create neighbor table NT
11       Compute N
12       NS = N + message iteration
13
14       rPower = random(1,3)
15
16       if( NS >= sizeNetwork + rPower || message iteration >= sizeNetwork + rPower )
17           Ask to save message and try to send it later.
18           stop everything
19
20       else
21
22          Select successors
23             succesorMatrix = ranking(NS,NT)
24          Forward message throw selected successors.
25             Send(succesorMatrix,message)
26          Waiting timer(message)
```

Figure 5.10: Utopia algorithm pseudo-code

Inside line 12 total number of successors is calculated, this number represents how many nodes will be finally chosen from the NT, it increases with each iteration to assure doing at least one broadcast before terminating this procedures, depends of computations performed by N function.

In line 14, rPower value is computed and assigned, as mention before this value represents a random number between a lower bound which tells at least how many complete floods needs to be done before stopping retransmission efforts and a higher bound which is the maximum number of complete floods that can be done.

Line 16 tells the source when to stop sending messages; it stops when message iteration or number of neighbors chosen reaches a value greater than the actual size of the network plus the rPower value explained before.

Finally, on line 26 message is pushed inside the waiting timer, this protocol controls the period of retransmissions after not receiving a confirmation message (RSReply) for each unique message.

5.6.2 Forward node protocol

If a message reaches a node which is not its destination, such node executes the following protocol. The purpose of it is to forward messages to the appropriated nodes accordingly with the power of the iteration, number of neighbors that the intermediary node has and size of the network.

This protocol maintains the same rules used when a source node creates and sends a message, but it doesn't maintain timers (therefore doesn't wait for

RSReply messages nor takes retransmission actions) and it's capable of generating an RSReply message in the case that the message is successfully forwarded to a destination.

It is also in charge of acting as a filter for destroying messages which time to live value has reached zero, this action is taken in order to stop message propagation to infinity.

Figure 5.11 shows pseudo code corresponding to this protocol. Important lines are marked in red and commented below.

```
1
2  forwardMessage(message)
3
4     Decrease message N_Hops
5        Message.nHops = Message.nHops - 1
6
7
8     If the destination (D) is neighbor
9        Send message directly
10          Send(D,message)
11       Send RSReply message
12          Send(S,RSReply)
13
14
15    Else if message number of hops <= 0
16       Stop everything
17    Else
18       Create neighbor table NT
19       Compute N
20       NS = N + Message Iteration
21       Select successors
22          succesorMatrix = ranking(NS,NT)
23          if succesorMatrix contains sourceNode or
24               predecessor remove it
25       Forward message throw selected successors.
26          Send(succesorMatrix,message)
```

Figure 5.11: Utopia algorithm pseudo-code 2

Line 5 code decrements by one the N_Hops value inside a message, suppose the case where N_hops isn't decreased, in such scenario the network quickly reach a saturation point, since messages keep traveling on continuously endless loops.

Finally, line 15 shows us when to stop forwarding messages, this is the case where n_hops has reach a value equal or below zero, it's possible to reach negative values since n_hops could be decreased by more than one in each hop by assigned a killingRate higher than 1 inside an specific node.

5.6.3 Waiting Timer

After a message is sent, if destination was found a RSReply message should be sent to the source node that originate the original message in order to know

that message was correctly delivered. If no confirmation is received retransmission of the message should take action, timer protocol defines how much time to wait until sending again another message and how to handle incoming confirmation messages.

Computing the waiting time is an important task, if it is too small network saturation could be quickly reach, in an opposite way if time is too large, long delays could be observed from both sides of the network (source and destination), therefore carefully choosing it is necessary in order to achieve better performance and efficiency.

How to choose the timer Considering the worst case scenario where a message need to travel through all the nodes of the network for delivering a message, and the same way back, plus some unpredictable temporarily loops timer is compute using the following formula:

$$TIMER = randomNumber\ (SizeNetwork*Delay,\ 2*SizeNetwork*Delay)$$

Where SizeNetwork represents the actual number of nodes participating on the network, and Delay is the time that one node k takes to forward a message. We can say that

$$1\ way = SizeNetwork*Delay$$

This is the time that takes a message to be delivered inside a worst case scenario.

Therefore timer formula can be reduced to:

$$TIMER = random\ number(1\ way,\ 2\ ways)$$

The delay introduce by one node k is directly proportional to the neighbors attached to him, since it needs to check using UFDD which nodes are within his range before actually forwarding the message. Therefore delay can be compute as follows:

$$Delay = averagedegradation(sizeOfNetwork) + Max\ pingTime$$

Where max ping time is the time that it takes to the slowest node to respond. Other way to define delay is: *delay is the time taken by UFDD when executed by one node.*

Average degradation is how fast delay time grows when the size of network is increase. This time is not equal to the sum of each node delay time.

As we can see timer is closely related with the size of the network and makes use of random values, this makes sense since if the network is bigger it will take more time to find and reach destination, random is used because we

don't know the exact configuration of the network in a specific point of time and actual destination could be any ware (really close to source or too far away).

Timer activation Each time that one new message is created one timer is assigned to it using the previous mention formula. Each time that timer is activated it resends the message using protocols described in previous sections. When a confirmation message is received and it matches the actual node, message is analyzed if there exist one timer that match such message it is removed in order to stop retransmissions for this message since now we know it was successfully deliver.

Figure 5.12 shows pseudo code corresponding to this protocol. Important lines are marked in red and commented below.

```
2  Waiting timer(message)
3
4  create message timer
5
6  if timer is activated
7
10       If RSReply not received for the message
11
12          message.N_hops = message.N_Hops + POWERHOPS
13          message.iteration = message.iteration + POWERITERATION
14          sendMessage(message)
15
16 if RSReply message recived AND RSReply.Source = Me
17       Alert user message was successfully sent
18       Stop timer
```

Figure 5.12: Utopia timer

Line 12 and 13 increase the depth search of the algorithm, since message wasn't found using initial depth power is increase, in order to perform such action two important settings are used. Powerhops and poweriteration, this two variables tells the algorithm how fast the depth should grow. By making use of this settings Utopia is capable of adapting to different environments and scenarios.

Finally, on line 18 message timer is stopped, this is done only in the case that a confirmation is received for such message; the other way that a message timer can be stopped is when message retransmission limit has been reached therefore too many attempts for reaching destination have done and non succeed, the number of attempts is defined inside source node protocol.

5.6.4 Number of neighbors

An important step while sending or forwarding a message is to select which neighbors will forward the message, but in order to do that first is necessary to

know how many node neighbors will be chosen for such purpose.

The value of the number of nodes selected is computed using the N Function; this function is in charge of returning a number accordingly with the following input values:

- Number of iteration
- Size of the network
- Number of neighbors
- Random values (in order to support some local load balancing)
- Other factors non considered yet

Therefore, such function is closely related also with the size of the network, actual number of neighbors and the iteration of current message. The relation between such factors and the output number is the real meaning of the function.

Utopia proposes three different selection functions (N function): percentage based on size of the network, evolutive function and finally the non function; descriptions of these are given on this document. Utopia functionality is not affected if selection function is changed to something unknown by the algorithm, any kind of function can be used as a selection function depending of the environments and possible scenarios.

Suppose the case that N function always returns as a value the size of the network, in this case Utopias algorithm is equivalent to a flood based algorithm with random retransmissions between 1 and 3.

Percentage based on network size In this case the number of neighbors taken on each iteration depends of size of the network and the number of iteration and percentage growing rate, an example with a 20% of growing rate looks as follow:
*N = sizeOfNetwork *(message iteration * numberNeighbors * porcentageGrowingRate)*
N = sizeOfnetwork (message iteration * numberNeighbors * 20%)*

What this example shows it's how the percentage of neighbors chosen on each iteration of the message is increased by 20%, this means that in a total of 5 iterations network will be completely flooded.

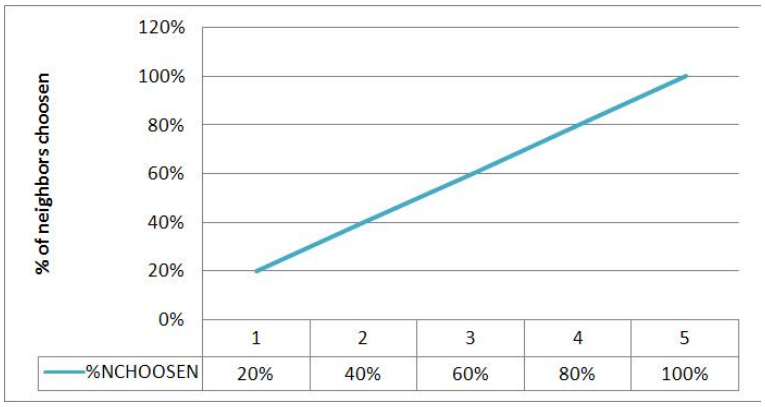


Figure 5.13: % of neighbors chosen

Figure 5.13 shows the result of plotting current iteration of the message against the percentage of neighbors chosen in such iteration (using a growing rate of 20% and a network size of 40 nodes), this function grows in a linear way, on iteration number five network is completely flooded, since each node running Utopia chose 100% of their corresponding neighbors.

Figure 5.14 corresponds to values plot in last graphic.

GROWING RATE	20%		
NETWORK SIZE	40		
ITERATION	%NCHOOSEN	NEIGHBORS	NCHOSEN
1	20%	31	6.2
2	40%	24	9.6
3	60%	32	19.2
4	80%	10	8
5	100%	30	30

Figure 5.14: % of neighbors chosen table

Since one node can have different number of neighbors at different points in time and the number of nodes from one node can be different from another, random values between 1 and the total size of the network where taken in order to simulate the dynamics of the network.

Figure 5.15 illustrates the behavior of two different nodes.

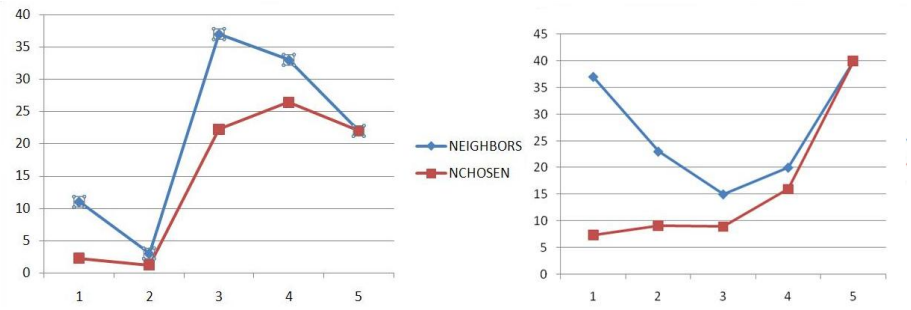


Figure 5.15: Comparison between neighbors choosen and actual neighbors

Graphics show how the number of neighbors chosen increase as the number of iterations grows, this means that each time that a message is re-send it uses more intermediary nodes until completely flooding the network.

Function N, only tells Utopia how fast to grow until completely flood the network and how to behave accordingly to different scenarios, taking into consideration as mention before, size of the network, iteration of message, and actual number of neighbors and sometimes using random values to introduce local balance and avoid making the same decisions all the time.

Utopias algorithm makes use of this linear percentage increasing function because of their simplistic nature. Percentage growing rate in a linear way doesn't perform perfectly, especially with large networks and complicated scenarios; it is used only as an easy example of how function N should be chosen.

Evolutive function This is a hypothetical case where feedback about a specific function can be obtain somehow (reveal by a magic oracle), what this kind of approach really does is to generate randomly different kind of functions and evaluate their performance (in real time) until a desire performance rate is achieved, and finally use the last N function that satisfies the performance rate.

Such situation is most not likely to happen inside any of our possible environments and scenario since evaluating the performance of N functions is a task difficult to accomplish and it's out of the scope of this project. In fact, there exist a large discussion of how to evaluate the performance of this kind of networks, since everything is distributed along all nodes on the net, and non entities on the net have a complete view of it, solving such problem is not as easy as it seems.

Even though this method is worth of mention since if a method for accurately analyze N functions performance on the fly is discovered, efficiency and

performance of Utopia could be increase in an amazing way.

Non function This is the case where no N function is given, equivalent of not telling the algorithm how to choose their neighbors, the non function will always return 0 as an output, and lucky Utopia can handle such situation, since if no n function is given Utopia implements the simplest case of number of neighbors selection, that is increasing by number of nodes selected on each iteration (N = message iteration).

Figure 5.16 shows a graphic which is the result of plotting the non function, it returns always value 0.

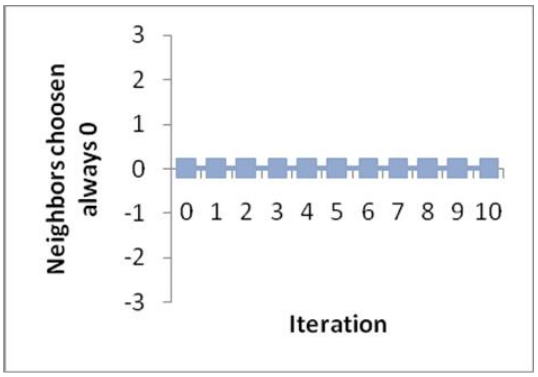


Figure 5.16: Non function plotting result

Function N Pseudo Code control Once the N function is set, obtaining number of neighbors is fair easy, figure 5.17 shows pseudo code corresponding to this protocol. Important lines are marked in red and commented below.

```
 2
 3 computeN(message iteration, numberNeighbors)
 4
 5 choose N function to use
 6
 7    case 1 (porcentageGrow)
 8
 9       return (message iteration * numberNeighbors * 20%)
10
11    case 2 (FUNCTION 2)
12
13    default
14       return 0
```

Figure 5.17: Funciont N pseudo code

Line 9 shows the implementation of percentage grows function, only one value is return proportionally to message iteration and size of the network. In the other hand, line 14 shows the non function which returns the constant value 0. Evolutive function is out of the scope of this document.

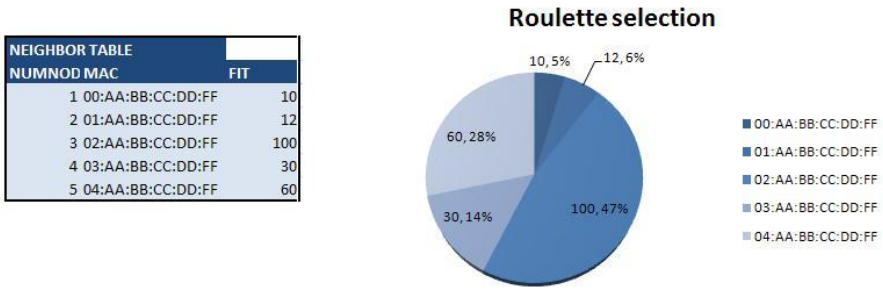
5.6.5 Neighbor selection protocol

The following section explains how successors are chosen when running Utopia algorithm, the idea is to fill successor matrix, for doing this is necessary to know how many nodes will be in such matrix (function N protocols is in charge of telling how many successors nodes are selected), possible successor nodes are selected according to their FIT. The better nodes are, the more chance to be selected they have.

Three selection processes are described, simple roulette selection, roulette selection with ranking and finally greedy or static selection process. The first two methods introduce some local load balancing because of their random nature when selecting successors, simple roulette selection introduce less local load balancing than ranking.

Roulette selection Imagine a roulette wheel where are placed all the neighbors of one node, each neighbor or possible successor has a one place on the roulette corresponding of how big is their FIT.

Figure 5.18 shows a roulette example where 5 nodes are participating in the election process.



NEIGHBOR TABLE		
NUMNOD	MAC	FIT
1	00:AA:BB:CC:DD:FF	10
2	01:AA:BB:CC:DD:FF	12
3	02:AA:BB:CC:DD:FF	100
4	03:AA:BB:CC:DD:FF	30
5	04:AA:BB:CC:DD:FF	60

Figure 5.18: Roulette example

The roulette starts to spin and selects one node. Successors with bigger FIT will be selected with a higher provability.

Pseudo code for roulette algorithm is the following [28]:

1. **[Sum]** Calculate sum of all nodes FIT in neighbor table - sum **N**
2. **[Select]** Generate random number from interval **(0,N) - r**
3. **[Loop]** Go through the neighbor table and sum fitnesses from **0 - sum s**. When the sum **s** is greater than **r**, stop and return the node where you are

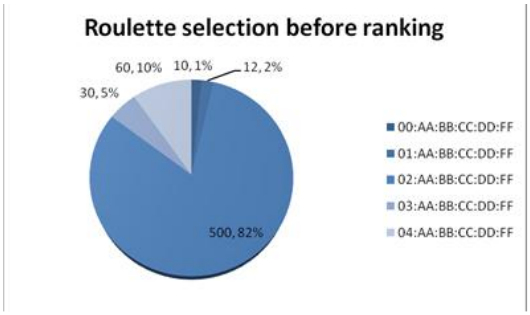
Of course, step 1 is performed only once for each neighbor table.

Rank Selection Roulette algorithm could have problems if FIT differs a lot between neighbors of one node, if the node with higher FIT has 95% of the roulette, then all the other nodes will have fewer chances to be selected [29].

In order to overcome such problem, rank selection is used, what it does is that before actually spinning the wheel neighbor table is ranked and each node receives a new FIT from this ranking.

Node with lowest FIT now will have a RANK of 1, the second lowest will have a RANK of 2, and so on, the higher FIT will now have a RANK of N, where N is the total number of neighbors (important it is NOT function N selected neighbors).

Figure 5.19 shows how node *02:AA:BB:CC:DD:FF* has higher probability to be elected compared with the rest of the nodes on the neighbor table.

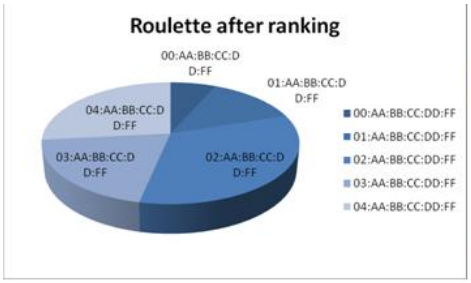


NEIGHBOR TABLE

NUMNODE	MAC	FIT
1	00:AA:BB:CC:DD:FF	10
2	01:AA:BB:CC:DD:FF	12
3	02:AA:BB:CC:DD:FF	500
4	03:AA:BB:CC:DD:FF	30
5	04:AA:BB:CC:DD:FF	60

Figure 5.19: Roulette before rankinng

After ranking, nodes with lower FIT now have more chances to be elected, having a consequence an increase on local load balancing.



NEIGHBOR TABLE

NUMNODE	MAC	FIT	RANK
1	00:AA:BB:CC:DD:FF	10	1
2	01:AA:BB:CC:DD:FF	12	2
3	02:AA:BB:CC:DD:FF	500	5
4	03:AA:BB:CC:DD:FF	30	3
5	04:AA:BB:CC:DD:FF	60	4

Figure 5.20: Roulette after rankinng

Static or greedy selection When using this method, instead of running roulette and apply ranking, successors are selected taking only the best ones from the neighbor table based on FIT until successor matrix is full. This selection process can perform better on environments where your target is not moving fast and there exist a lot of static Bluetooth devices like printers. Since message propagation is done through better class devices that don't move delivery of the message is faster and more accurate.

This particular kind of method can lead to lose local load balancing introduce by the other methods and random decisions. Choosing which selection process to use depends highly of the environment. Utopia algorithm runs using roulette ranking selection since it tries to perform as best as possible in all types of scenarios and environments but other selection methods can be use without disrupting the rest of the functions of it.

Chapter 6

Application

UtopiaChat is a first implementation of Utopia's algorithm; it is a distributed chat application that allows a limited amount of users to exchange messages between them along large distances by using intermediary Bluetooth devices.

This chapter covers the following topics:

- User manual
- Tests

6.1 User manual

This section explains how to use Utopia's chat and presents a summarize guide of its functionality. First minimum requirements to run it are mention and then a detailed description of each function.

6.1.1 Requirements

In order to run UtopiasChat the following requirements must be fulfilled by the target device:

- Device must have some type of Bluetooth communication device (class1, class2 or class3)
- Target device should count with a J2ME virtual machine capable of running CDLC configurations

UtopiasChat is able to run over any Java Virtual machine capable of using CDLC configurations, please refer to your mobile device, PDA, laptop or printer manual for specifications of how to install java applications for your specific kind of device.

When starting the application for the very first time follows these steps: First enable Bluetooth connections on the device and then introduce initial configuration settings.

Enabling Bluetooth connections In order to enable Bluetooth connectivity go to connections and introduce enable Bluetooth (for more information please refer to device configuration manual). It's important to perform this step since UtopiaChat Bluetooth has to be on in order to work properly.

Figure 6.1 shows a screenshot of a normal configuration Bluetooth being use by UtopiaChat.

Figure 6.1: Configuration Bluetooth

In figure 6.1 shows that the user has to insert the following data:

- Bluetoot: indicates the current state of Bluetooth
- My phone's vibility: Indicates the visibility of the device
- My phone's name: indicates the username field in Utopia chat

Input initial configuration settings To introduce the setting data the user has to follow these steps:

1. Start SSE1 application. Select from the mobile phone menu. The initial screen is shown (figure 6.2-a)
2. Select Intro and the main screen is shown (figure 6.2-b)
3. Select Options
4. Choose Set Configuration. The data device screen is shown (figure 6.2-c)
5. Introduce data device
    - Type device: indicate printer, mobile phone, laptop, PDA or computer
    - Class device: 1, 2 or 3
    - Phone number

Figure 6.2: Mobile phone screenshots - setting data device

6.1.2 Actions

In this section is explained the actions that can be doing with UtopiaChat and how perform its.

The User list window is the main screen with the list of employees (figure 6.3). From this window it's possible to perform different actions such as send a message, add employee, remove employee, modify setting and exit.

Figure 6.3: Employees' list

Send message

The following steps need to be done in order to send a message to another user.

1. Go to employees' list window (figure 6.4-a)
2. Select the employee which has to receives the message
3. Write message. Once the destination user is select, a new screen which will give the opportunity to write the message will be opened (figure 6.4-b)
4. Select send
5. Click OK. An alert message which ask if the user is sure to send the message is shown (figure 6.4-c)

Figure 6.4: Send message screenshots

Depending of the status of the message sent, different confirmation screens will be presented:

1. Directly deliver. Is shown when the message has been sent directly (figure 6.5-a)

2. Message processed. An alert message informs that message as been processed, becuase the destination is not in the Bluetooth range as it is shown (figure 6.5-b). In order to deliver this message the application uses Utopia algorothm

After sending the message, it will appear in the screen as it shown in figure 6.5-c

Figure 6.5: Alert's messages

Once the message has been sent the chat Form is shown to the user (figure 6.6).

Figure 6.6: Chat form

Add Employee

This action allows users to include another new employee into the list of employees. To perform these actions the user has to follow these steps:

1. Go to employees' list window (figure 6.7-a)
2. Select options. (figure 6.7-b)
3. Select add employee.

The application will start a device discovery. While the device discovery run on the mobile phone the screenshot of figure 6.7-c is shown on the screen. Once the search is finished a new list of employees is uptdated (figure 6.7-d).

Figure 6.7: Add emplyee screenshots

Remove employee

Remove employee is used when an employee leaves the environment forever. This user has to delete him from his employees' list. To perform this action the user has to follow these steps:

1. Go to the employees' list window (figure 6.8-a)
2. Go to the employee which has to be deleted in the list
3. Select options (figure 6.8-b)
4. Select remove employee

After this action employee window is shown with the updated list (figure 6.8-c).

Figure 6.8: Remove employee screenshots

View chat

This action shows current conversations established with different users. to perform this action the user has to follow these steps:

1. Go to the employees' list window (figure 6.9-a)
2. Select options (figure 6.9-b)
3. Select View Chat

After these steps the chat Form is shown (figure 6.9-c).

Figure 6.9: View chat screenshots

Receive a message

When a user receives messages, an alert inform about this event (Figure 6.10-a) and the chat Form display will appear(figure 6.10-b).

Figure 6.10: Receive a message screenshots

Exit

To leave the application the user has to go to the main window, employees' list window and select exit

6.2 Tests

Tests are related with direct communication between one device and its neighbours. They serve as a premise for the tests done to that check the complete behaviour of various nodes.

During the implementation of our application we have done several tests on it. At beginning we test our application on the emulator, and later on real mobile phones. We were trying to figure out if the application was behaving as we expected.

Tests are related with direct communication between one device and its neighbours. They serve as a premise for the tests done to that check the complete behaviour of various nodes.

The first test that we have done regards the discovery device; if we change the Major code on the mobile phone will the device be discovered by other mobile phones?

We changed the number 0x200 (Phones) with 0x600 (Printers) and 0x100 (Laptops, PDAs) and we noticed that the mobile phone didn't discover the other devices.

After that we tested the service discovery process, so we changed some number in the UUID number or we changed the name of the service, also in this case our test was successfully because the mobile phone didn't discover the other mobile phones which we changed the service. Then we run a lot of mobile phones on the emulator and we were sending messages from one device to all of them to understand if the application was able to work with a lot of mobile phones and connects with the right service. We repeat also this test on 4 real mobile phones and our test was successful too.

Next we tested to send message to some device when they were not in his range anymore. On the Emulator we were running 2 mobile phones, and after doing the device discovery we shut down one of the 2 mobile phones, after that the other were sending a message to the one which has been shut down. We received an error alert which shows to the user that the other mobile phone is not anymore in the range Bluetooth.

We test it also in real mobile phones, the first time we were shouting down one of the mobile phone and sending a message, in the second time we went far from his range Bluetooth and sending a message, in both cases we got the same result as on the emulator. This was the behaviour that we had expected.

In figure 6.11 this test is shown.

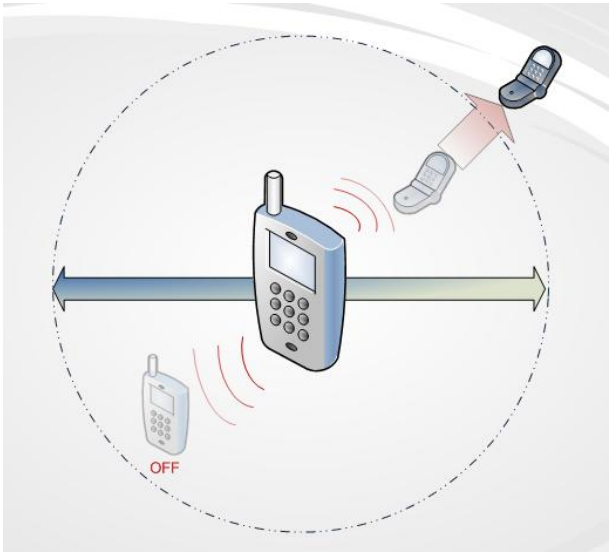


Figure 6.11: Out of range

After showing the error message we delete the service record of the mobile phone which is not anymore in the range Bluetooth. We tested this process printing on the screen the service which is still available. Also in this test our application behaved as we have expected.

While we were testing our application we find out that sometimes we didn't manage to show all the devices that were in the range Bluetooth. After several tests we understood that the thread which is in charge to do the service discovery terminate without finishing to inspect the current remote device. So it doesn't add the service found to the list of the services. We resolved this problem using a wait method each time a service discovery starts and a notify method each time the service discovery ends. For more details see section 4.2.2.

In the following paragraphs, more tests are done to check if the application behaves as expected.

- Fast device discovery
- Forward a message with one intermediary node
- Forward a message with several intermediariers node
- Concurrent reception of messages
- Modifications in the total network table (add, modify and remove)

6.2.1 Fast device discovery

One technique that allows the application to work faster is the fast device discovery (UFDD). The fast discovery is used inside Utopia algorithm to find which nodes are in the range. UFDD was tested as separated module before actually using it inside Utopia algorithm. First we tested the typical device discovery to find 3 mobile phones and later the UFDD was tested. The time spent by UFDD was noticeably smaller than the one spent by the typical device discovery. The test was successful and we noticed that it is very important to use this technique in order to get faster behaviour of our application.

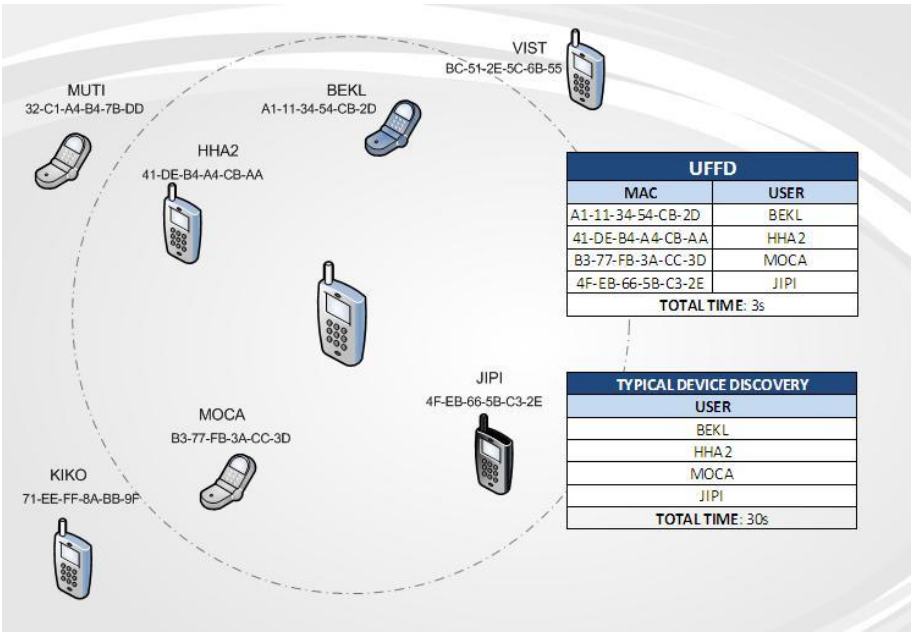


Figure 6.12: Typical Bluetooth discovery vs UFDD

In figure 6.12 mobile located at the center of circle starts a normal device discovery and time taken to perform such task is registered. Later the UFFD device discovery is done and time is also measure. Result of using these two techniques can be observed looking at tables inside the picture.

The only difference between them is the time required by these two different devides discovery techniques. Tables show that while the typical device discovery required around 30 seconds to find the Bluetooth devices located in its range, the UFFD Bluetooth discovery required only 3 seconds to find them.

The test was successful and we noticed that it would be very important to use this technique to get a faster behaviour of our application.

6.2.2 Forward message with one intermediary node

Figure 6.13 shows an example where the message is received through an intermediary node.

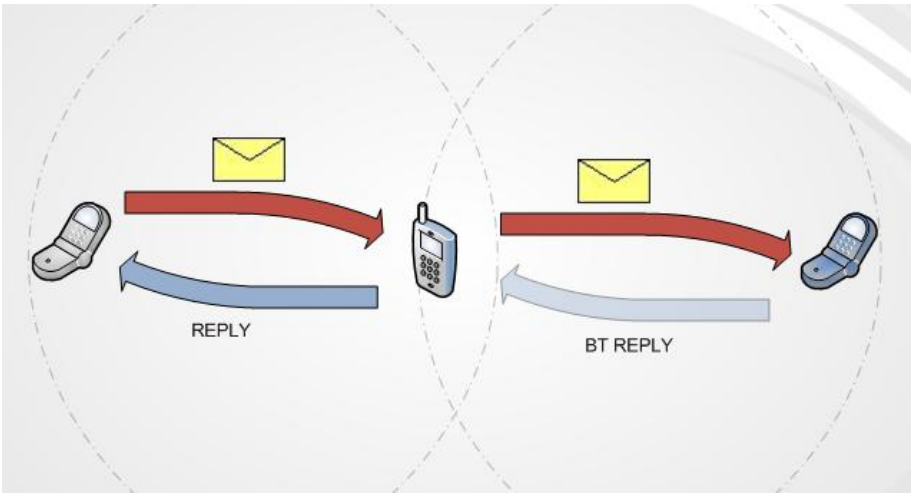


Figure 6.13: Forward message with one intermediary node

The first step to check if it is possible to send a message through several intermediary nodes is obviously testing the forward of a message with one intermediary node. The test started with the source node doing the device discovery and it found only the intermediary node. Next, the intermediary node did its own device discovery and it found both the source node and the destiny node. Then the source node chose the destination from the employees´ list and sent a message to it. Finally, the destination received the message and the source node received the confirmation from the intermediary node. Everything worked with an acceptable time thanks to the UFDD Bluetooth discovery.

6.2.3 Forward a message with several intermediariers node

In order to observe Utopias confirmation message broadcast nature the following test was done, seven mobile phones were participating on this test, source node S which wants to send a message to destination D, 4 intermediary nodes connected to the source node, and one last intermediary node which is connected to destination D (figure 6.15).

Figure 6.14 describes the input values to the system and the expected behavior.

INPUT VALUES	EXPECTED RESULTS	RESULT
Configuration of the network show in the graphic and message.	Message is delivered to destination D and confirmation message is shown on source node.	Test was successful, message was received and confirmation show into a reasonable time frame.
Notes Stage one, RSMessage is initially forward only to two destinations due Roulette protocol and message iteration 0; stage 2, message is forwarded again to last successor; stage 3, last successor deliver message successfully and creates RSReply message; stage 4, RSReply message is broadcasted using 4 intermediare nodes since RSReply messages use a max iteration level (complete flood of the network)		

Figure 6.14: Forward a message with several intermediariers node table

Figure 6.15 shows the network topology of this test, and message forwarding decisions taken.

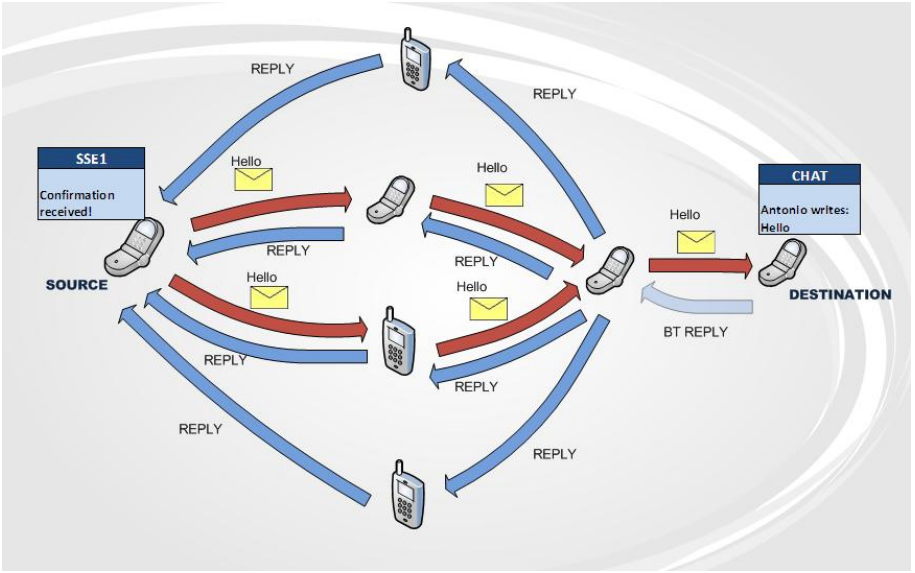


Figure 6.15: Forward a message with several intermediariers node scenario

6.2.4 Concurrent reception of messages

The application was tested by sending 2 messages to the same destination from different sources to check if one server was able to deal with several messages at the same time. The result of this test was that the 2 messages were received by the destination and displayed in the chat screen. Also, both sources received the confirmation. This test shows the correct behaviour of using par-

allel threads in the application.

This test is shown in figure 6.16.

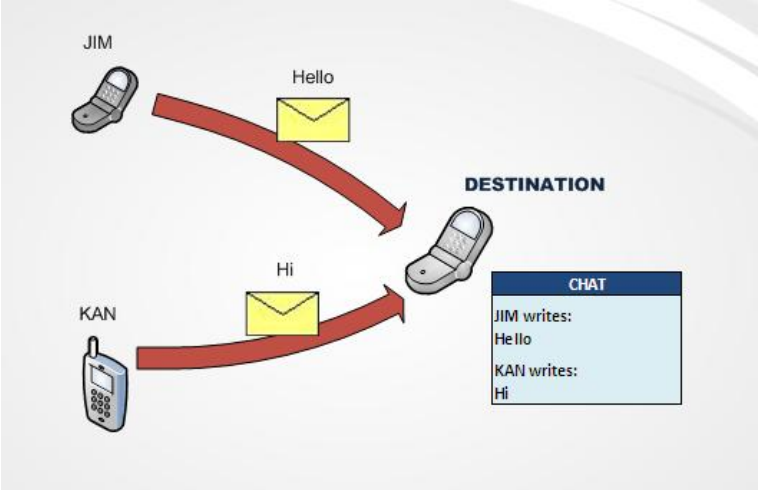


Figure 6.16: Server listener

In figure 6.16 two different source: Jim and Kan send a message to the same destination. As we can see in this figure, both messages are displayed in the chat screen of the destination employee.

6.2.5 Modifications in the network (add, modify and remove nodes)

This test checks the behaviour of the user table. Situations like when a new user is inserted, when a user leaves the environment or when one user modifies its characteristics are included. Observe the table of figure 6.17.

Username	MAC Address	Class	Type	Phone Number
Beatriz	E2-3F-5A-3D-5A-4B	1	printer	03888892
Daniel	1B-5C-8A-1B-5E-2E	2	laptop	45667521
Antonio	4D-5B-5E-6A-7B-1D	2	mobile phone	48943011

Figure 6.17: Default user table

Add user When a new user is hired by one environment that uses Utopia is included inside its user table. This process is done by one device that executes the typical device and service discoveries and then the new device and all its characteristics are inserted into the database. In Figure 6.18 a new row has been inserted into database.

Username	MAC Address	Class	Type	Phone Number
Beatriz	E2-3F-5A-3D-5A-4B	1	printer	03888892
Daniel	1B-5C-8A-1B-5E-2E	2	laptop	45667521
Antonio	4D-5B-5E-6A-7B-1D	2	mobile phone	48943011
Ruben	5A-EE-4D-5D-CB-9C	1	mobile phone	78532109

Figure 6.18: User table: add user

Modify user When one user starts the application, he has to introduce his type of mobility, class of device and phone number as it is shown in figure 6.19.

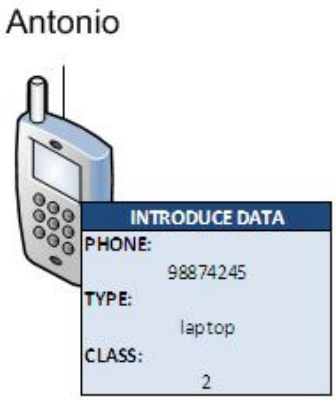


Figure 6.19: Modify employee state

Let's assume that some of this data will change, for instance the user will change the phone number, and each time that another user starts the normal device discovery data of the user which change the phone number will be updated.

In figure 6.20, a Bluetooth device changes its type, in fact the username Daniel was a Laptop in figure 6.17. After changing its type to a Printer type, if the other users start a normal device discovery, type of Daniel device will be updated in the database of the device that started the device discovery, as it is shown in Figfure 6.20

Username	MAC Address	Class	Type	Phone Number
Beatriz	E2-3F-5A-3D-5A-4B	1	printer	03888892
Daniel	1B-5C-8A-1B-5E-2E	2	laptop	45667521
Antonio	4D-5B-5E-6A-7B-1D	2	laptop	98874245
Ruben	5A-EE-4D-5D-CB-9C	1	mobile phone	78532109

Figure 6.20: User table: modify user

Remove user When one user is hired out of the environment it is deleted from the database and its data are not available anymore. In the test, one user was deleted from the database and later the UFDD was done with the device deleted inside the range (even with the application running in it) and the mentioned device was not found. Figure 6.21 shows the resulting database without the old user *Daniel*.

Username	MAC Address	Class	Type	Phone Number
Beatriz	E2-3F-5A-3D-5A-4B	1	printer	03888892
Antonio	4D-5B-5E-6A-7B-1D	2	laptop	98874245
Ruben	5A-EE-4D-5D-CB-9C	1	mobile phone	78532109

Figure 6.21: User table: remove user

Chapter 7

Conclusions and future work

Different solutions to the Bluetooth range extending problem have been presented; Utopia is the ultimate solution for such purpose, it creates a Bluetooth ad hoc high dynamic network by making use of different technologies, philosophies and strategies.

A distributed chat application demonstrates the power of Utopia, by using it, is possible to send messages to Bluetooth enable devices separated by long distances. Such task is not easy and do have some limitations.

One of the new techniques introduces by Utopias algorithm is making use of intelligent random decisions, some nodes have more chance to be elected during the selection process, and therefore some local load balancing is introduced having as a consequence distribution of overall network bandwidth between them. In brief, random decisions assure that not always the same actions are taken and therefore dealing with dynamics of the network.

There is no single unit responsible for all operations when sending a message using Utopia algorithms, each node can perform equally right without the need of a central unit coordinating all actions. Utopia is fault tolerance and decentralized, such characteristics outperforms most Bluetooth chat systems.

In the other hand Bluetooth range is increased by making use of intermediary Bluetooth devices.

High dynamics of the network were a big trouble since the beginning of Utopias design due the same nature of the network, Utopias team realizes that assuring message delivery inside unstructured pure p2p networks (mixed with an ad hoc approach) it's almost impossible, node failures are impossible to predict and exist too many join and departures of nodes. Utopia makes their best effort to assure message delivery but no complete warranty of message delivery is given until a confirmation message is receive.

Real large scale environments tests are difficult to perform, since equipment for such tests is expensive and large quantities of Bluetooth devices are not always available; this is the case of Utopias teams hence test were only done

through small quantities of nodes.

Utopias do have some limitations worth of mention and suffer of some scalability problems. In order to perform a fast discovery every device has to subscribe to the service, this means that all Bluetooth devices need to be known before actually sending messages and correctly updated each time a new node is added to the network.

UFDD delay grows really slow as the number of nodes in the network increase since total network table is scanned using concurrent threads in parallel therefore saving time. A strong advantage of UFDD over a normal device discover is that when a device is doing a normal device discovery the rest of the devices can't find such device, UFDD overcomes this problem.

If a normal device discovery is used instead of UFDD while running Utopia, a lot of nodes may never receive messages becasue of the normal discovery limitations. Moreover, time spent during this process will be considerably much higher than UFDD since each node of the network needs to build the neighbor table by making use of the device discovery.

Bluetooth has many advantages but when we talk about large scale implementations do have some limitations (besides range), such as messages failure when consecutively sending messages to the same node (even if they are really close). BT also has some problems to communicate depending the distance between the source and next successor; when devices are close the transfer is faster than if they are far away of each other.

This study can still be improved in some aspects. The approaches proposed mainly rely on a fixed size of network and UFDD to enhance broadcast accuracy and reduce overhead; however list of users could be updated dynamically and be shared among all the nodes of network, if UFDD technique is modified to run even faster overall performance of the system could highly increase. In brief, if we can explore the network more precisely and efficiently against the dynamic nature of P2P networks, we can improve the broadcast performance.

Chapter 8

Ackowledgements

Utopias teams deeply thanks Rene Hansen for actively participate on Utopias development as an advisor, his guidance lead the team to achieve higher objectives.

Bibliography

[1] NTRG. *P2P Networks.* http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/Intro.html. 2007.

[2] Department of Computer Science University of Maryland. *Analysis and Comparison of P2P Search Methods.* http://www.ieee.org/portal/site. 2007.

[3] Institute of Computer Science and Applied Mathematics University of Bern. *BLR: Beacon-Less Routing Algorithm for Mobile Ad-Hoc Network.* www.mics.org/getDoc.php?docid=520&docnum=1. 2007.

[4] Stitut national de techerne en informatique et an automatique. *Link State Routing in Wireless Ad-Hoc Networks.* ftp://ftp.inria.fr/INRIA/publication/publi-ps-gz/RR/RR-4874.ps.gz. 2007.

[5] Mercadolibre. *Bluetooth vs infrared.* http://guia.mercadolibre.com.ve/bluetooth-vs-infrarojo-caracteristicas-basicas-3526-VGP. 2006.

[6] Wikipedia. *Bluetooth.* http://en.wikipedia.org/wiki/Bluetooth.

[7] Projector. *Bluetooth Review.* http://www.metu.edu.tr/ e125037/projector/belgeler/Bluetooth_Review.pdf.

[8] Symbian Developer Network. *Symbian Academy.* http://developer.symbian.com/main/academy/.

[9] Roberto Perdomo. *Python, an agile programming language.* http://maracay.velug.org.ve/descargas/PonenciaPython.pdf.

[10] TodoSymbian. *Comparison of symbian C++ and J2ME.* http://www.todosymbian.com/postt30150.html.

[11] Bluetooth official webpage. *How Bluetooth technology works.* http://www.bluetooth.com/Bluetooth/Learn/Works/.

[12] Urs Steiner. *J2ME: Introduction, Configurations and Profiles.* http://www.ifi.unizh.ch/ riedl/lectures/Java2001-j2me.pdf. 2001.

[13] Riggs Taivalsaari Peursem Huopaniemi Patel Uotila. *Programming Wireless Devices with the Java 2 Platform Micro Edition.* http://www.informit.com/content/images/0321197984/samplechapter/riggsch02.pdf.

[14] Eric Giguere C. Enrique Ortiz. *Mobile Information Device Profile for Java 2 ME*. 2005.

[15] Sun Microsystems official webpage. *Mobile Information Device Profile (MIDP)*. http://java.sun.com/products/midp/overview.html.

[16] Ross Lee Graham. *Peer to peer: Toward a Definition*. http://www.ida.liu.se/conferences/p2p/p2p2001/p2pwhatis.html.

[17] SearchNetworking.com. *Peer to peer*. http://searchnetworking.techtarget.com/sDefinition/ 0,,sid7_gci212769,00.html.

[18] Wikipedia. *Peer to peer*. http://en.wikipedia.org/w/index.php?title=Peer-to-peer&oldid=176915801.

[19] Wapedia. *Peer to peer*. http://wapedia.mobi/es/Peer-to-peer?t=8.

[20] Wikipedia. *Peer to peer*. http://es.wikipedia.org/wiki/Peer-to-peer#Clasificaci.C3.B3n.

[21] Rijubrata Bhaumik. *Peer to Peer content delivery in the Internet*. http://www.tml.tkk.fi/Opinnot/T-110.6120/2007/fall/p2p.pdf.

[22] Jukka K. Nurmien. *Content Search Unstructured P2P*. http://www.tml.tkk.fi/Opinnot/T-110.7100/2007/P2P_070919_2.pdf. 2006.

[23] Coulouris Dollimore Kindberg. *Distributed Systems- concepts and design*. 2005.

[24] Dejan S. Milojicic. *Peer to peer computing*. http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.pdf.

[25] Martin de Jode. *Programming Java 2 Micro Edition on Symbian OS*. 2006.

[26] Tommi Mikkonen. *Programming Mobile Devices*. 2007.

[27] LeCroy. *Bluetooth*. http://www.lecroy.com/tm/Solutions/Protocol/ bluetooth.asp?menuid=29.

[28] University of Applied Sciences. *Genetic Algorithms* http://cs.felk.cvut.cz/xobitko/ga/.

[29] Adam Marczyk. *Genetic Algorithms and Evolutionary Computation*. http://www.talkorigins.org/faqs/genalg/genalg.html. 2004.

List of Figures

1.1 Distribution of stationary points 6
1.2 Flood scenario 8
1.3 Possible network structure 9

2.1 Wireless technologies comparison 12
2.2 Develop platforms comparison 13
2.3 Classes of Bluetooth chips 14
2.4 Bluetooth piconet and scatternet scenarios 15
2.5 Java's family 16
2.6 Examples of devices for each edition 16
2.7 Layers of J2ME 17
2.8 J2ME configurations 18
2.9 Configuration characteristics 18
2.10 J2ME Architecture 19
2.11 Benefits of MIDP 20
2.12 Packages of MIDP API 21
2.13 Centralized P2P network architecture 24
2.14 Pure P2P network architecture 25
2.15 Hybrid P2P network architecture 26
2.16 P2P's architectures advantages and disadvantages 27

3.1 Direct message scenario 30
3.2 Direct message scenario 2 30
3.3 Send message through one or more intermediaries and get the confirmation message scenario 31
3.4 Send message through one or more intermediaries and loose the confirmation message scenario 32
3.5 Destination node not reachable scenario 33
3.6 Use cases 34
3.7 Description Use cases 34
3.8 Class Diagram 35

4.1 Architecture 39
4.2 Server's steps 40
4.3 ServerClass class code 41
4.4 ServerClass class code 2 42
4.5 Server's behavior 43
4.6 Client's requests 44
4.7 Client's steps 45

4.8 Discovery Listener . . . 46
4.9 Device discovered method . . . 47
4.10 Major device classes . . . 47
4.11 Discovery device behaviour . . . 48
4.12 Service Discovery Listener . . . 49
4.13 startServiceSearch and serviceSearchCompleted methods . . . 50
4.14 Service Discovered method . . . 51
4.15 Service Discovery behaviour . . . 51
4.16 Print job method . . . 53

5.1 Different kind of nodes . . . 55
5.2 Mombile phone database . . . 56
5.3 RS structure - forward message . . . 58
5.4 RS structure - direct message . . . 59
5.5 RSReply structure - forward reply . . . 59
5.6 RSReply structure - direct reply . . . 59
5.7 Two way handshakes - scenario A . . . 60
5.8 Two way handshakes - scenario B . . . 61
5.9 Message life cycle . . . 62
5.10 Utopia algorithm pseudo-code . . . 64
5.11 Utopia algorithm pseudo-code 2 . . . 65
5.12 Utopia timer . . . 67
5.13 % of neighbors chosen . . . 69
5.14 % of neighbors chosen table . . . 69
5.15 Comparison between neighbors choosen and actual neighbors . . . 70
5.16 Non function plotting result . . . 71
5.17 Funciont N pseudo code . . . 72
5.18 Roulette example . . . 73
5.19 Roulette before rankinng . . . 74
5.20 Roulette after rankinng . . . 74

6.1 Configuration Bluetooth . . . 76
6.2 Mobile phone screenshots - setting data device . . . 77
6.3 Employees' list . . . 78
6.4 Send message screenshots . . . 79
6.5 Alert's messages . . . 80
6.6 Chat form . . . 80
6.7 Add emplyee screenshots . . . 81
6.8 Remove employee screenshots . . . 82
6.9 View chat screenshots . . . 83
6.10 Receive a message screenshots . . . 84
6.11 Out of range . . . 86
6.12 Typical Bluetooth discovery vs UFDD . . . 87
6.13 Forward message with one intermediary node . . . 88
6.14 Forward a message with several intermediariers node table . . . 89
6.15 Forward a message with several intermediariers node scenario . . . 89
6.16 Server listener . . . 90
6.17 Default user table . . . 90
6.18 User table: add user . . . 91
6.19 Modify employee state . . . 91

6.20 User table: modify user 92
6.21 User table: remove user 92